IBM Research

The IBM High Performance Computing Toolkit

I-Hsin Chung
Advanced Computing Technology
IBM T.J. Watson Research Center
ihchung@us.ibm.com

What is it?

- **IBM long-term goal:**
  - An automatic performance tuning framework
    - Assist users to identify performance problems
    - Provide possible solutions
  - A common application performance analysis environment across all HPC platforms
  - Look at all aspects of performance (communication, memory, processor, I/O, etc) from within a single interface
- **Where we are: one consolidated package**
  - One consolidate package (AIX, Linux/Power)
  - Tools for MPI, OMP, processor, memory etc
  - Operate on the binary and yet provide reports in terms of source-level symbols
  - Dynamically activate/deactivate data collection and change what information to collect
  - One common visualization GUI

IBM HPC Toolkit Software Components

- **Hardware (CPU) Performance**
  - Xprofiler
  - HPM Toolkit
    - hpmcount
    - libhpm
    - hpmstat
- **Shared Memory Performance**
  - Dpomp, PompProf
- **Message-Passing Performance**
  - MPI profiler/tracer
- **Memory Performance**
  - Sigma
- **Performance Visualization**
  - PeekPerf
- **I/O Performance**
  - MIO (modular I/O)

Supported Platforms

- **AIX**
- **Linux**
  - PowerPC
  - Blue Gene /L and Blue Gene /P
  - Intel x86 (coming soon)
  - AMD (coming soon)
- **Windows (Intel/AMD) + Mac (coming soon)**
  - Offline Peekperf visualization capability only

AGENDA

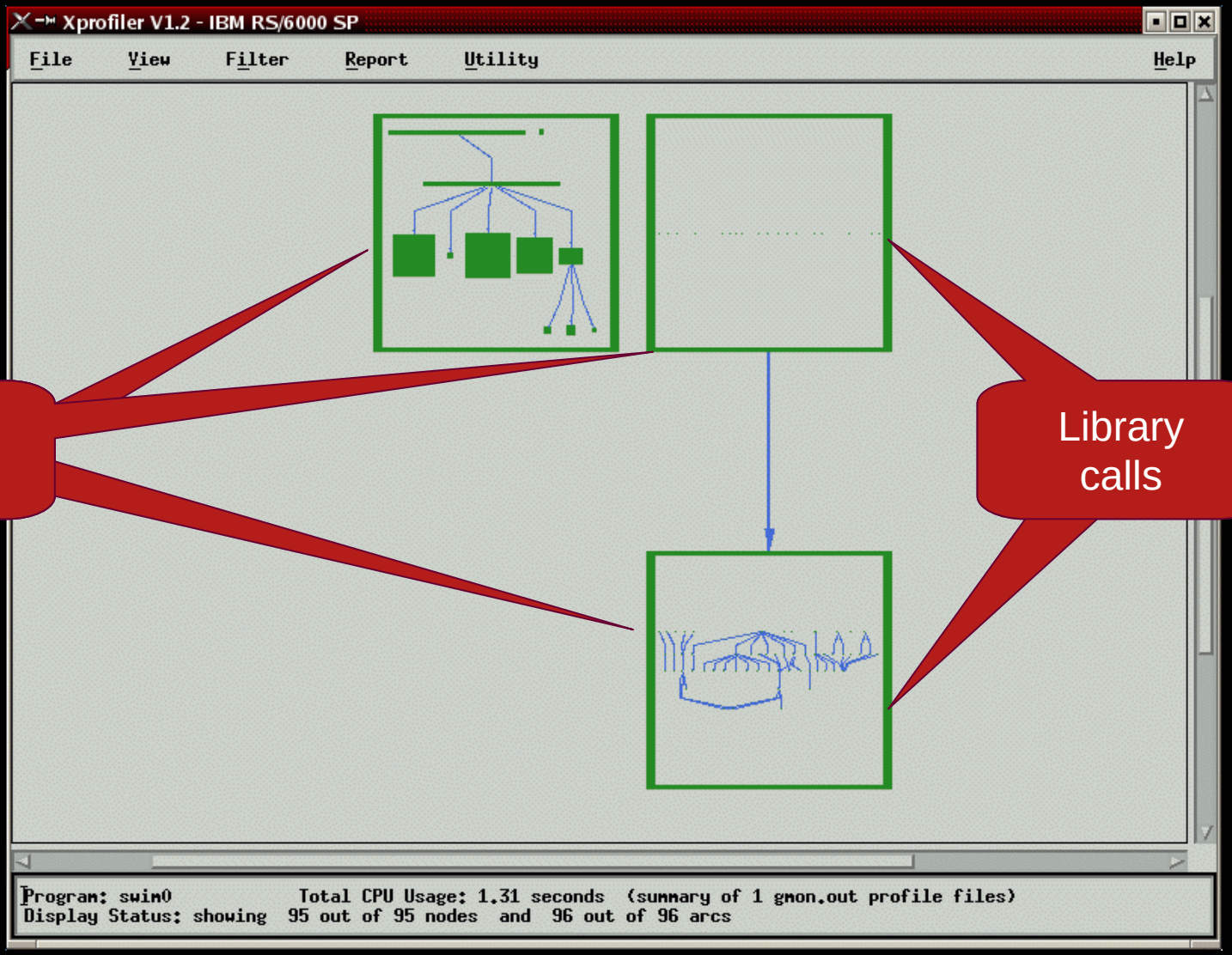
- **Xprofiler: call-graph profiling**
- **HPM: hardware counter data**
- **MPI Profiler/Tracer: MPI profiling**
- **PompProf: OpenMP profiling**
- **SIGMA: memory profiling**
- **MIO: I/O profiling and optimization**
- **IBM HPC Toolkit**
- **Questions/Comments**

XProfiler

Xprofiler

- **Visualizer CPU time profiling data**
- **Compile and link with -g -pg flags + optimization**
- **Code execution generates gmon.out file**
  - MPI applications generate gmon.out.1, …, gmon.out.n
- **Analyze gmon.out file with Xprofiler**
  - xprofiler a.out gmon.out
- **Important factors:**
  - On AIX time-sampling interval is 0.01 sec
  - Profiling introduces overhead due to function calls

Xprofiler - Initial View

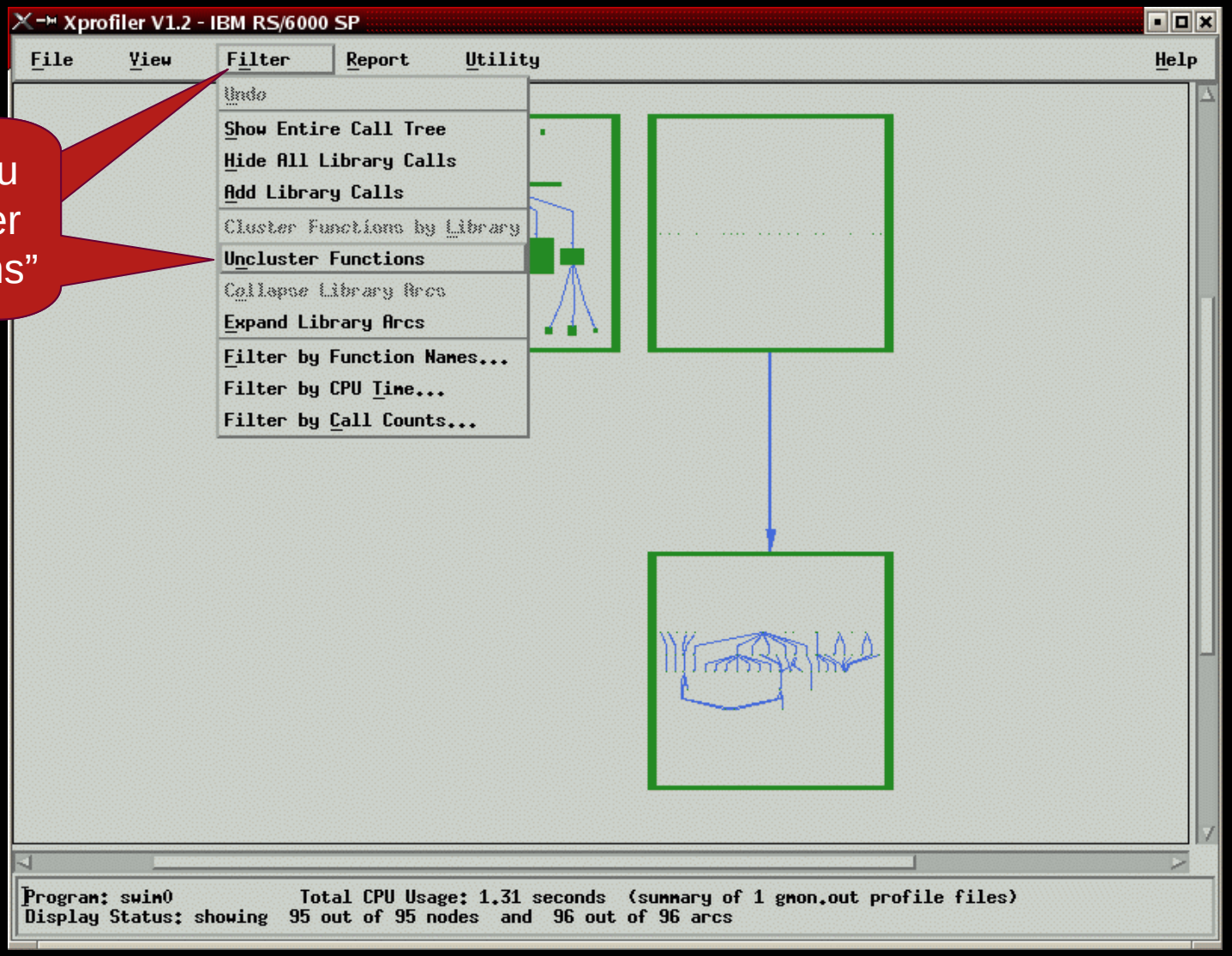


Clustered functions

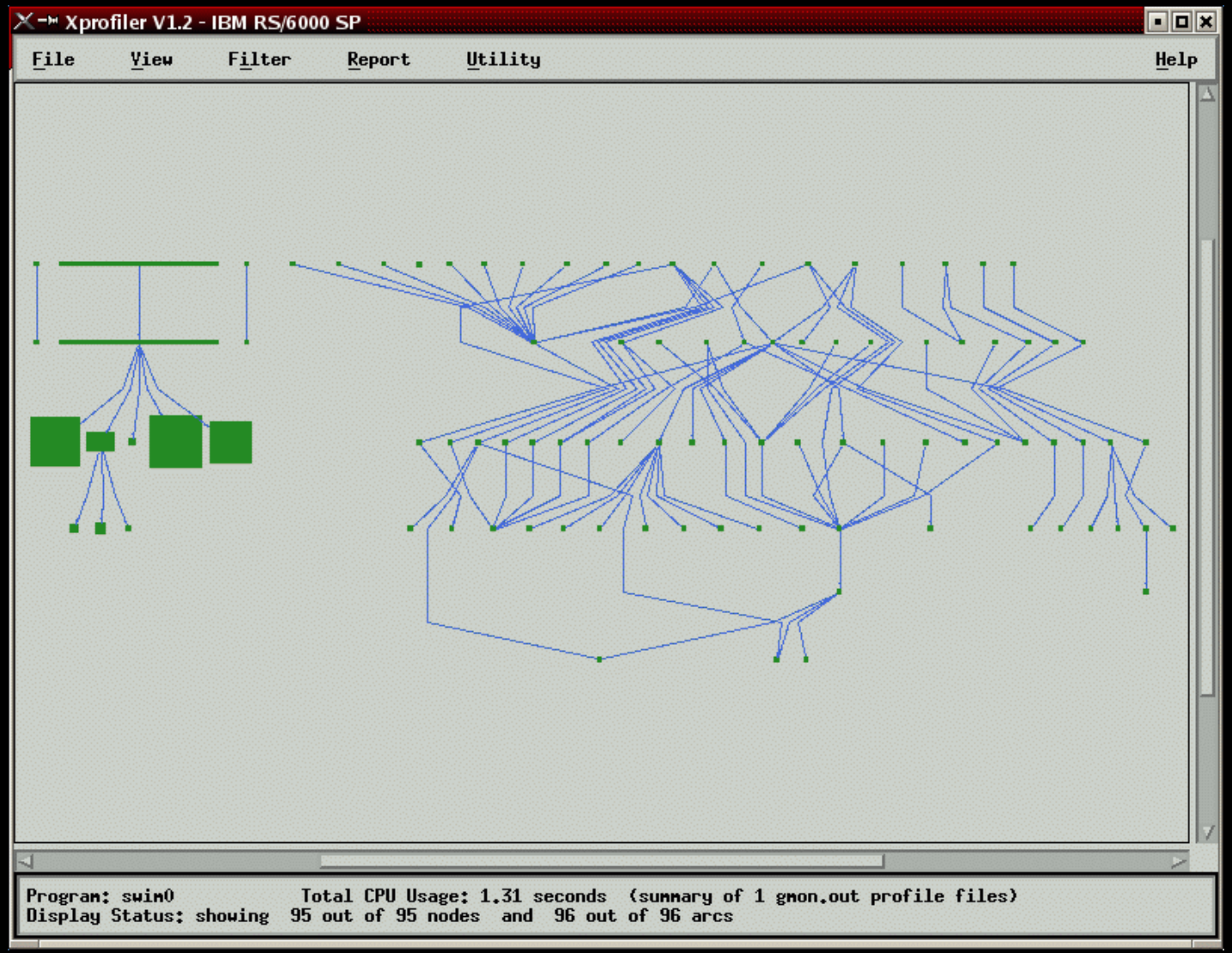
Library calls

Xprofiler - Unclustering Functions

on “Filter” menu select “Uncluster Functions”



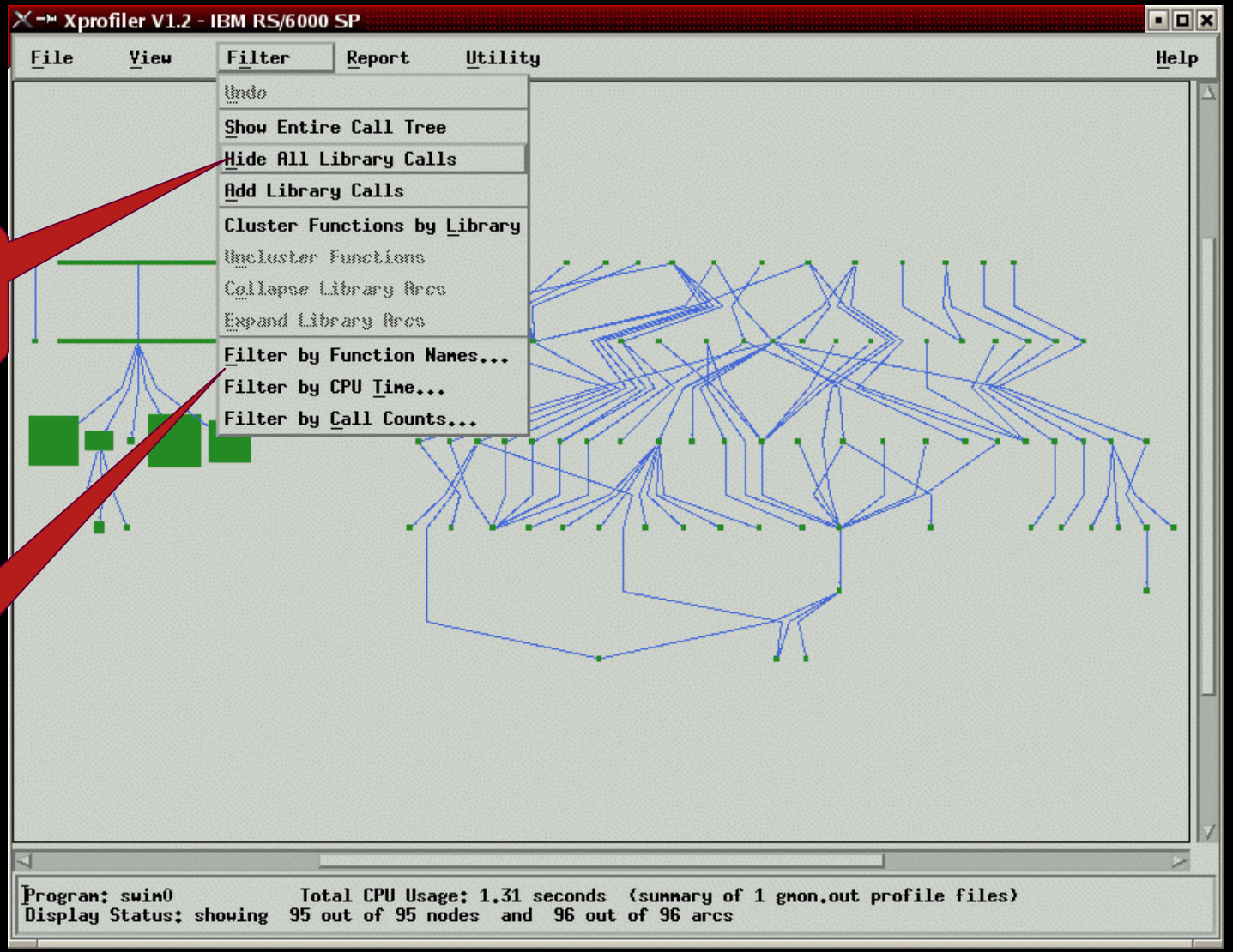
Xprofiler - Full View - Application and Library Calls



Xprofiler - Hide Lib Calls Menu

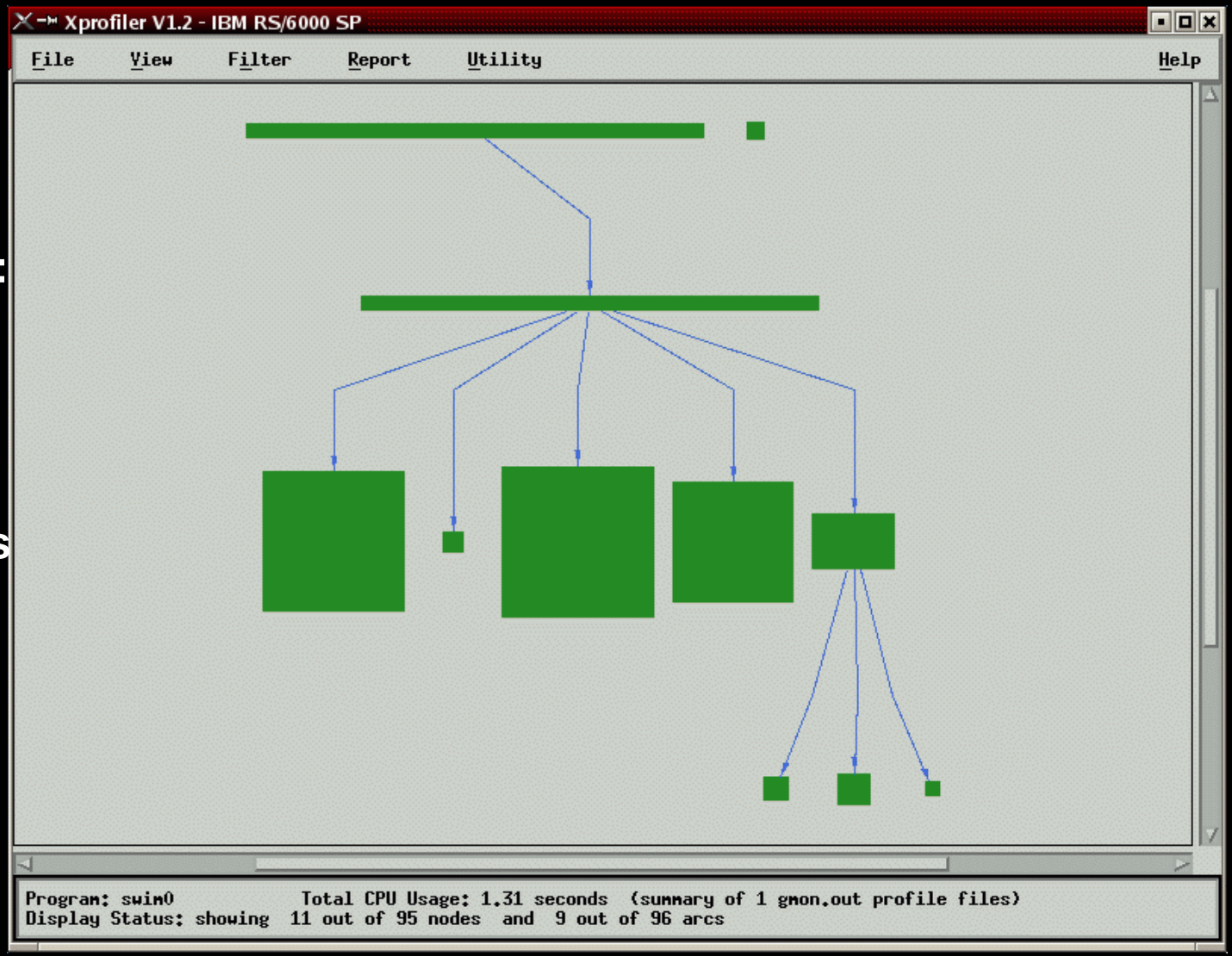
Now select "Hide All Library Calls"

Can also filter by: Function Names, CPU Time, Call Counts

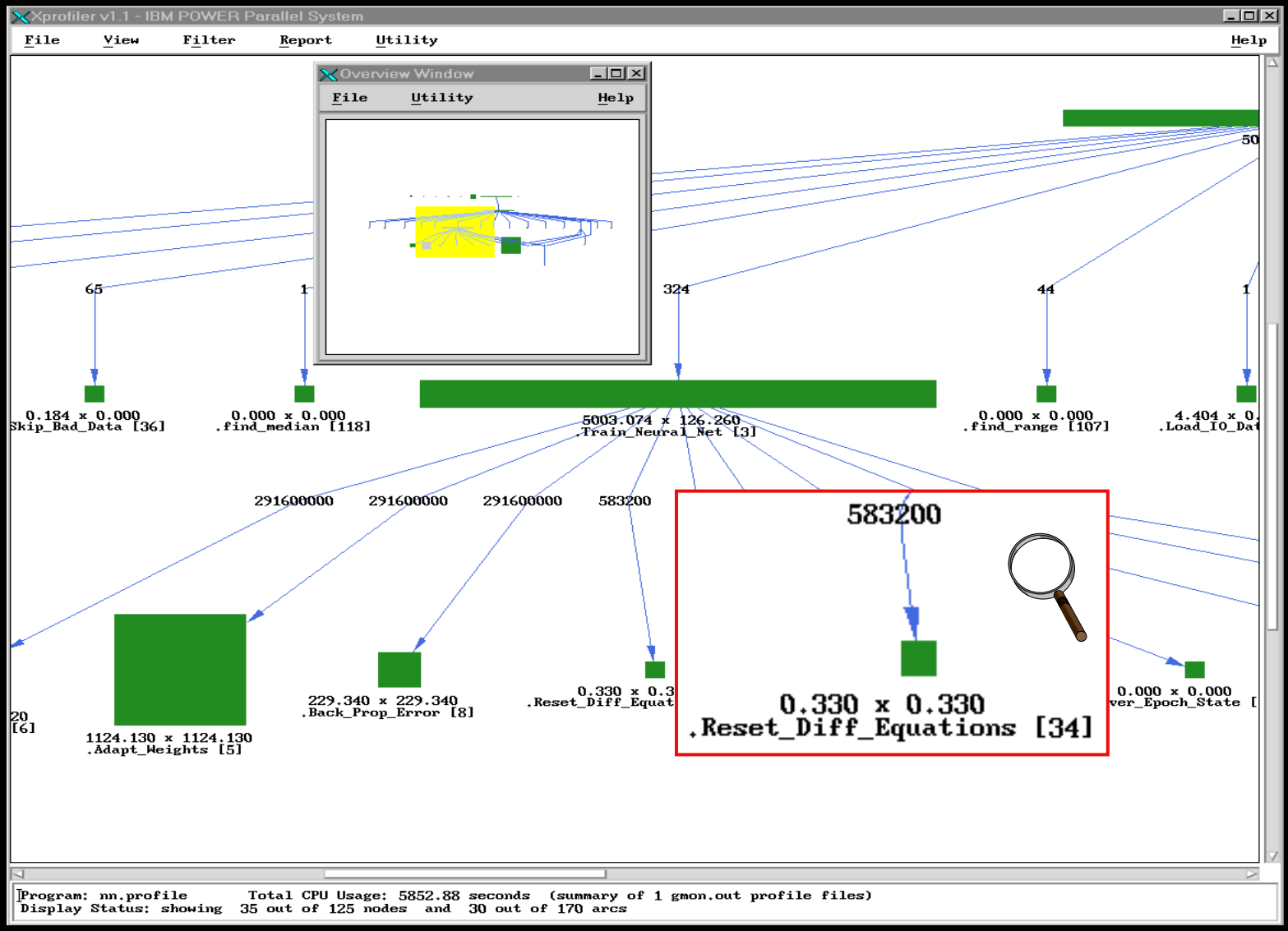


Xprofiler - Application View

- **Width of a bar: time including called routines**
- **Height of a bar: time excluding called routines**
- **Call arrows labeled with number of calls**
- **Overview window for easy navigation (View → Overview)**

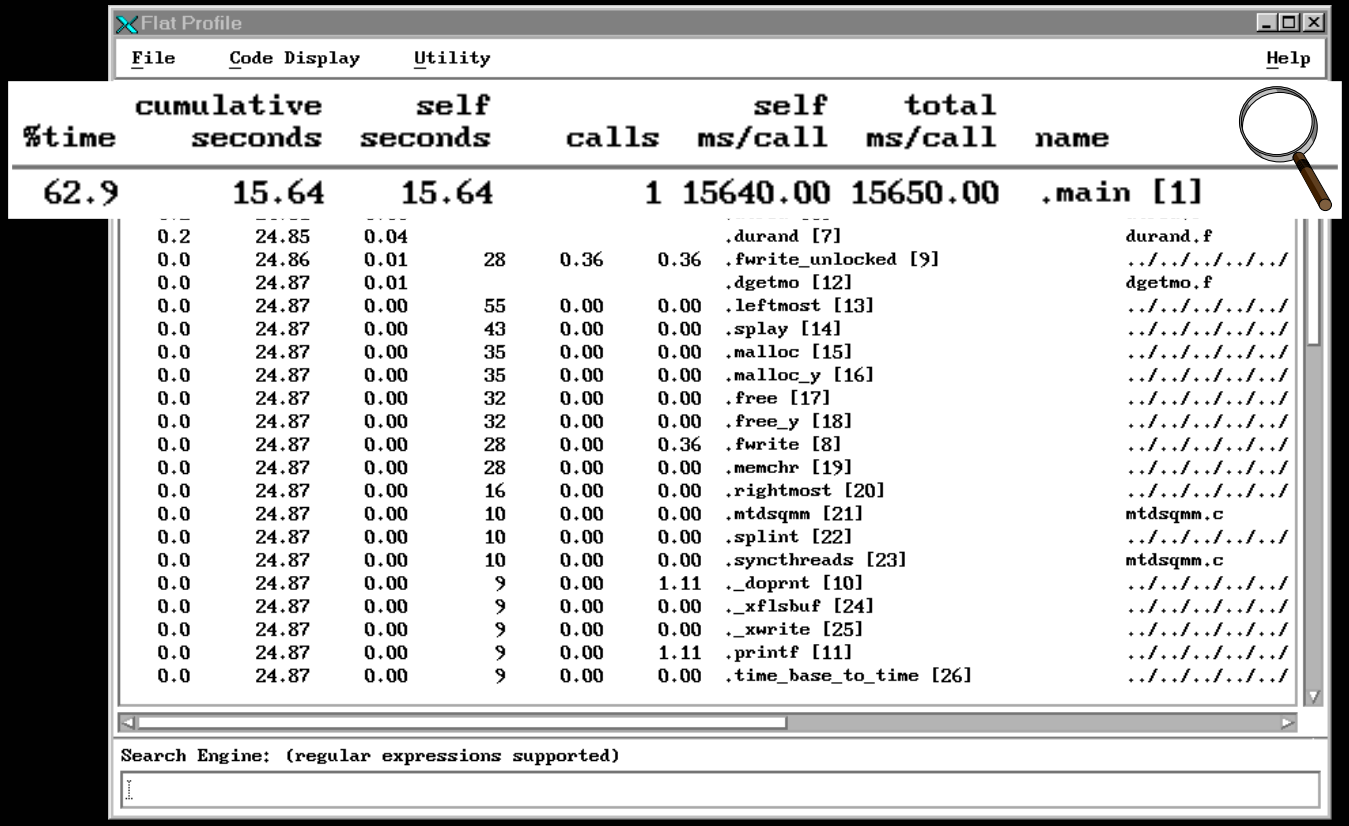


Xprofiler: Zoom In



Xprofiler: Flat Profile

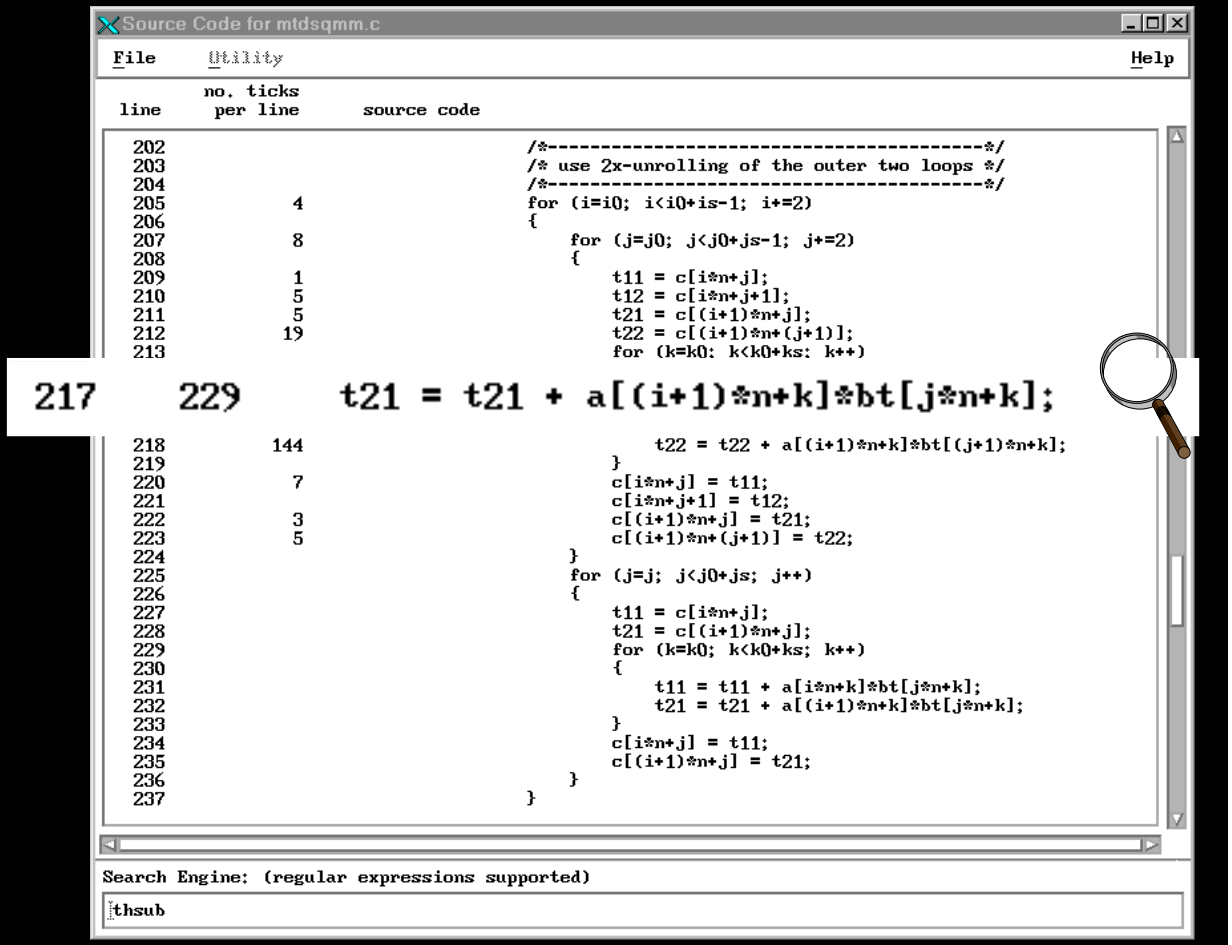
- **Menu `Report` provides usual gprof reports plus some extra ones**
  - Flat Profile
  - Call Graph Profile
  - Function Index
  - Function Call Summary
  - Library Statistics



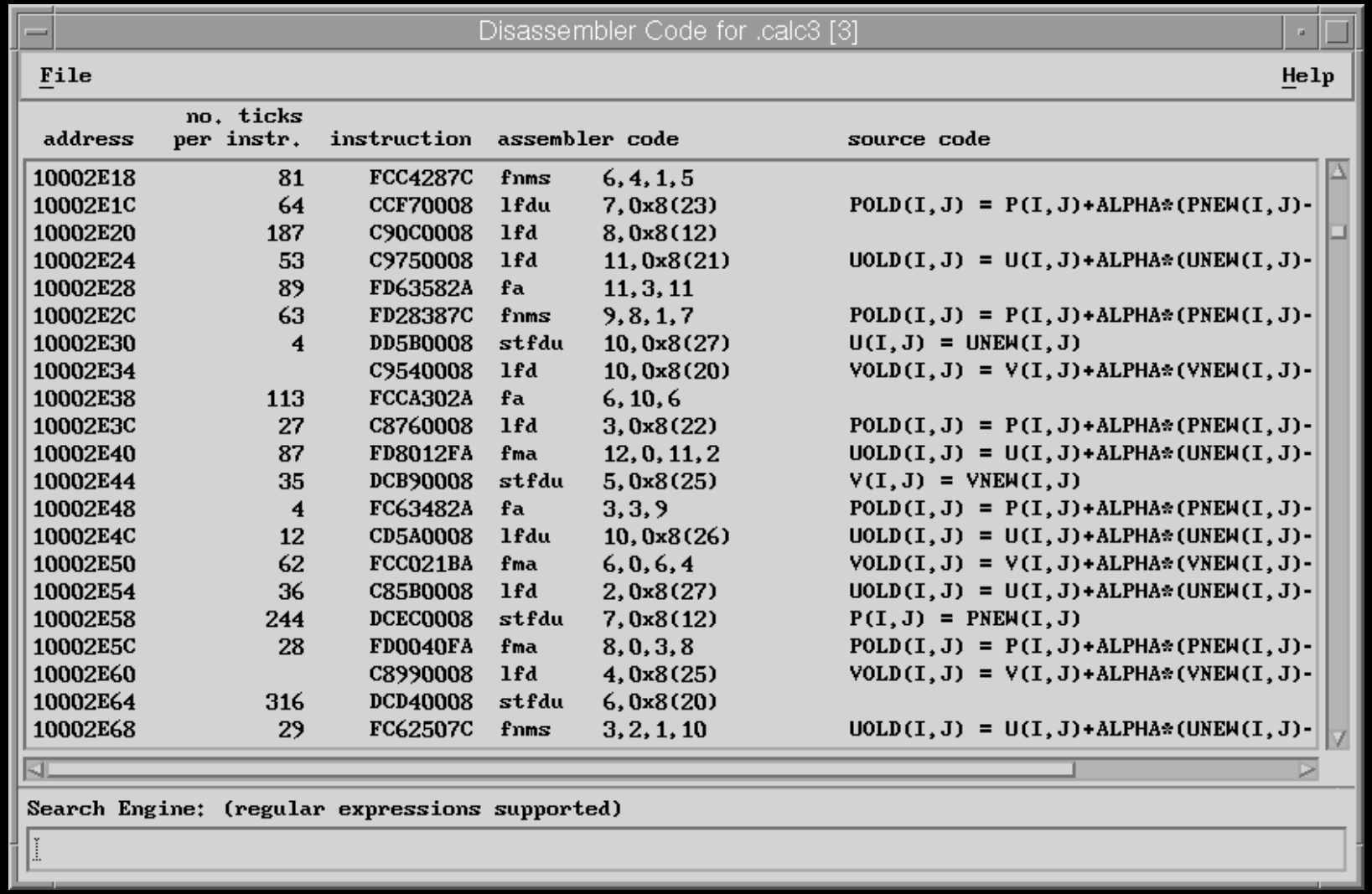
Xprofiler: Source Code Window

- **Source code window displays source code with time profile (in ticks=0.01 sec)**

- **Access**
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



Xprofiler - Disassembler Code



Xprofiler: Tips and Hints

- Simplest when gmon.out.*, executable, and source code are in one directory
  - Select "**Set File Search Path**" on "**File**" menu to set source directory when source, and executable are not in the same directory
  - Can use **-qfullpath** to encode the path of the source files into the binary
- By default, call tree in main display is "clustered"
  - Menu Filter → Uncluster Functions
  - Menu Filter → Hide All Library Calls
- Libraries must match across systems!
  - on measurement nodes
  - on workstation used for display!
- Must sample realistic problem (sampling rate is 1/100 sec)

HPM

HPM: What Are Performance Counters

- **Extra logic inserted in the processor to count specific events**
- **Updated at every cycle**
- **Strengths:**
  - Non-intrusive
  - Very accurate
  - Low overhead
- **Weakness**
  - Provides only hard counts
  - Specific for each processor
  - Access is not well documented
  - Lack of standard and documentation on what is counted

HPM: Hardware Counters

- **8 counters on PPC970 and Power4, 6 counters on Power 5/5+**
  - Several (100+) events per counter
- **48 UPC counters on Blue Gene/L, 328 events**
- **256 UPC counters on Blue Gene/P, ~1000 events**

- **Events can not be selected independently**
  - PPC970: 41 groups, default: 23
  - Power 4: 64 groups, default: 40
  - Power 5: 140 groups (AIX 5.2), 148 groups (AIX 5.3), default 137
  - Power 5+: 152 groups, default 145
  - Blue Gene/L: 16 groups

- **hpmcount –l or –c for list of counters**

HPM: Hardware Counters

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions

- Useful derived metrics
  - **IPC - instructions per cycle**
  - **Float point rate (Mflip/s)**
  - **Computation intensity**
  - **Instructions per load/store**
  - **Load/stores per cache miss**
  - **Cache hit rate**
  - **Loads per load miss**
  - **Stores per store miss**
  - **Loads per TLB miss**
  - **Branches mispredicted %**

- **Derived metrics allow users to correlate the behavior of the application to one or more of the hardware components**
- **One can define threshold values acceptable for metrics and take actions regarding program optimization when values are below the threshold**

HPM Toolkit

- **Data capture, analysis, and presentation of hardware performance metrics for HPC applications and systems**
  - **hpmcount**
    - Starts a program and at the end of the execution provides a summary with hardware counters information and derived metrics
      - Simple to use, no source code modification
  - **libhpm**
    - Instrumentation library for performance measurement of Fortran, C, and C++ applications
  - **hpmstat**
    - Hardware monitoring at system level

HMPCOUNT Usage

- **[poe] hpmcount [-o <name>] [-u] [-n] [-x] [-g <group>] <program>**
- **hpmcount [-h] [-l] [-c]**
  - <program> program to be executed
  - -h displays this help message
  - -o <name> output file name
  - -u make the file name <name> unique
  - -n no hpmcount output in stdout when "-o" flag is used
  - -x adds formulas for derived metrics
  - -g <group> PM_API group number
  - -l list groups
  - -c list counters and events

HPM: Environment Variables

- **HPM_EVENT_SET**
- **HPM_VIZ_OUTPUT**
- **HPM_OUTPUT_NAME**
- **Memory, cache, and TLB miss latencies:**
  - HPM_TLB_LATENCY
  - HPM_MEM_LATENCY
  - HPM_L3_LATENCY
  - HPM_L35_LATENCY
  - HPM_L2_LATENCY
  - HPM_L25_LATENCY
  - HPM_L275_LATENCY
- **...**

HPMCOUNT Output - Resource Usage

```
Total execution time (wall clock time): 25.329646 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 21.500000 seconds
Total amount of time in system mode          : 1.820000 seconds
Maximum resident set size                    : 25444 Kbytes
Average shared memory use in text segment    : 77688 Kbytes*sec
Average unshared memory use in data segment  : 54386768 Kbytes*sec
Number of page faults without I/O activity   : 6372
Number of page faults with I/O activity      : 8
Number of times process was swapped out      : 0
Number of times file system performed INPUT  : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                  : 0
Number of IPC messages received              : 0
Number of signals delivered                  : 1
Number of voluntary context switches         : 168
Number of involuntary context switches       : 2605
```

HPMCOUNT Output - Group 5

```
PM_DATA_FROM_L3 (Data loaded from L3)                    :        64164220
PM_DATA_FROM_MEM (Data loaded from memory)               :         5623627
PM_DATA_FROM_L35 (Data loaded from L3.5)                 :          281896
PM_DATA_FROM_L2 (Data loaded from L2)                    :       947542929
PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared)      :             739
PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared)    :             567
PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified)  :             903
PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified)    :             120

Memory traffic                                  :        2879.297 MBytes
Memory bandwidth                                :          96.317 MBytes/sec
Estimated latency from loads from memory        :           1.730 sec
Total loads from L3                             :          64.446 M
L3 traffic                                      :        8249.103 MBytes
L3 bandwidth                                    :         275.945 MBytes/sec
Estimated latency from loads from L3            :           5.067 sec
L3 Load miss rate                               :           8.026 %
Total loads from L2                             :         947.545 M
L2 traffic                                      :      121285.793 MBytes
L2 bandwidth                                    :        4057.200 MBytes/sec
Estimated latency from loads from L2            :           8.747 sec
L2 Load miss rate                               :           6.886 %
```

Estimation based on user input

HPMCOUNT Output - Group 60

```
PM_FPU_FDIV (FPU executed FDIV instruction)              :       262675952
PM_FPU_FMA (FPU executed multiply-add instruction)       :      4466499631
PM_FPU0_FIN (FPU0 produced a result)                     :      6857269712
PM_FPU1_FIN (FPU1 produced a result)                     :      9200414352
PM_CYC (Processor cycles)                                :    152475196089
PM_FPU_STF (FPU executed store instruction)              :      3430135817
PM_INST_CMPL (Instructions completed)                    :    117862217347
PM_LSU_LDF (LSU executed Floating Point load instruction) :     17921080598

Utilization rate                         :          99.658 %
Load and store operations                :       21351.216 M
Instructions per load/store              :           5.520
MIPS                                     :        1001.455
Instructions per cycle                   :           0.773
HW Float point instructions per Cycle    :           0.105
Floating point instructions + FMAs (flips) :       17094.048 M
Flip rate (flips / WCT)                  :         145.245 Mflip/sec
Flips / user time                        :         145.743 Mflip/sec
Weighted Floating Point Instructions     :       20771.511 M
Weighted flip rate                       :         176.492 M Wflip/sec
FMA percentage                           :          52.258 %
Computation intensity                    :           0.801
```

Measurement of system activity
User time / WCT

Measurement of FPU & LSU utilization:
Flips / LS

Derived Metrics

- Utilization rate
- Total FP load and store operations
- MIPS
- Instructions per cycle/run cycle/load store
- % Instructions dispatched that completed
- Fixed point operations per Cycle or load/stores
- Branches mispredicted percentage
- number of loads per load miss
- number of stores per store miss
- number of load/stores per L1 miss
- L1 cache hit rate
- number of loads per TLB miss
- number of loads/stores per TLB miss

- Total Loads from L2
- L2 load traffic
- L2 load bandwidth per processor
- Estimated latency from loads from L2
- % loads from L2 per cycle
- Total Loads from local L2
- local L2 load traffic
- local L2 load bandwidth per processor
- Estimated latency from loads from local L2
- % loads from local L2 per cycle
- Total Loads from L3
- L3 load traffic
- L3 load bandwidth per processor
- Estimated latency from loads from L3
- …

LIBHPM

- **Allows to go in the source code and instrument different sections independently**
- **Supports Fortran, C, and C++**
- **For each instrumented section provides:**
  - Total count & duration (wall clock time)
  - Hardware performance counters information
  - Derived metrics
- **Provides resource usage statistics for the total execution of the instrumented program**
- **Supports:**
  - MPI, OpenMP, & pThreads
  - Multiple instrumentation points
  - Nested instrumentation
  - Multiple calls to an instrumented point

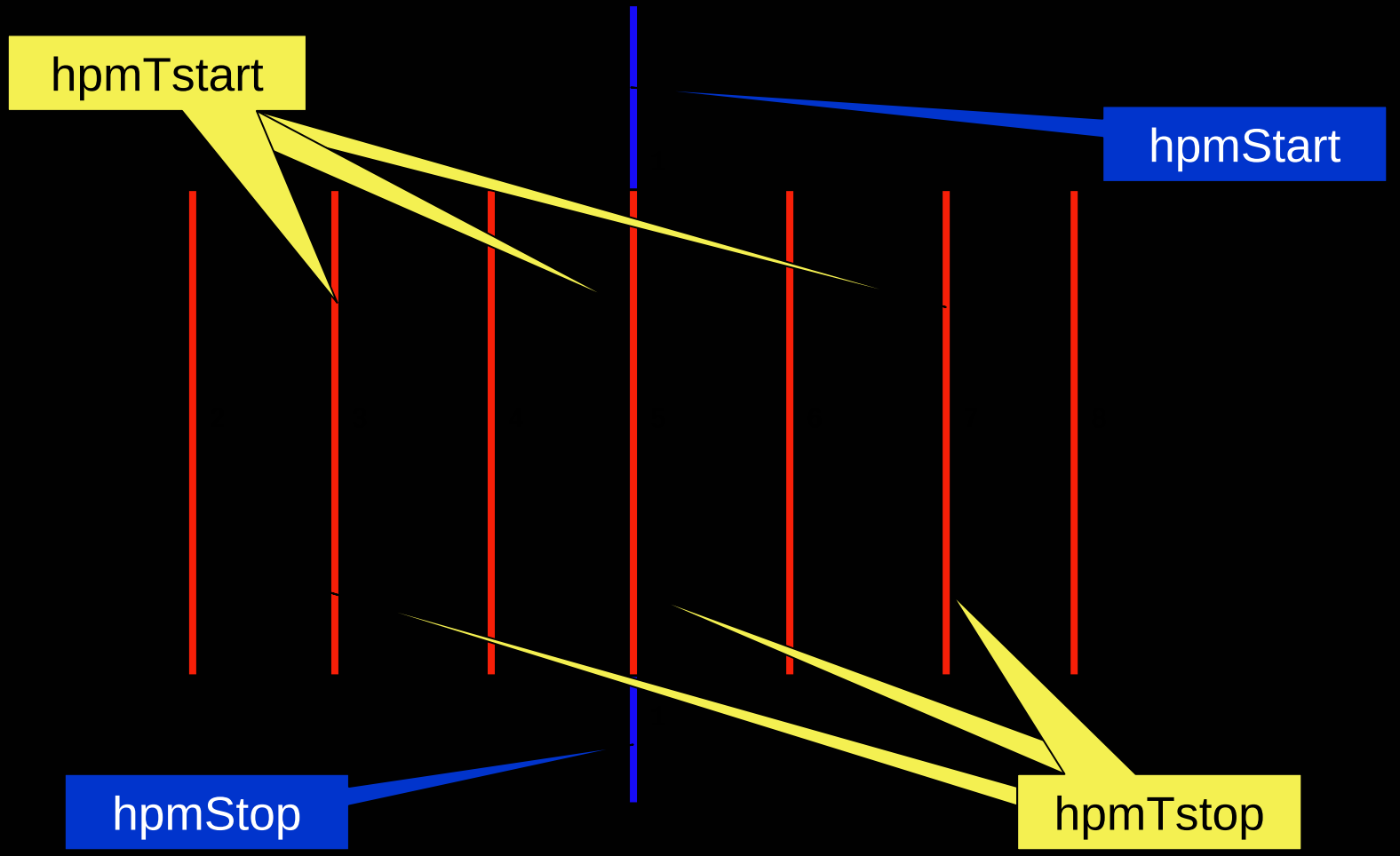
Using LIBHPM

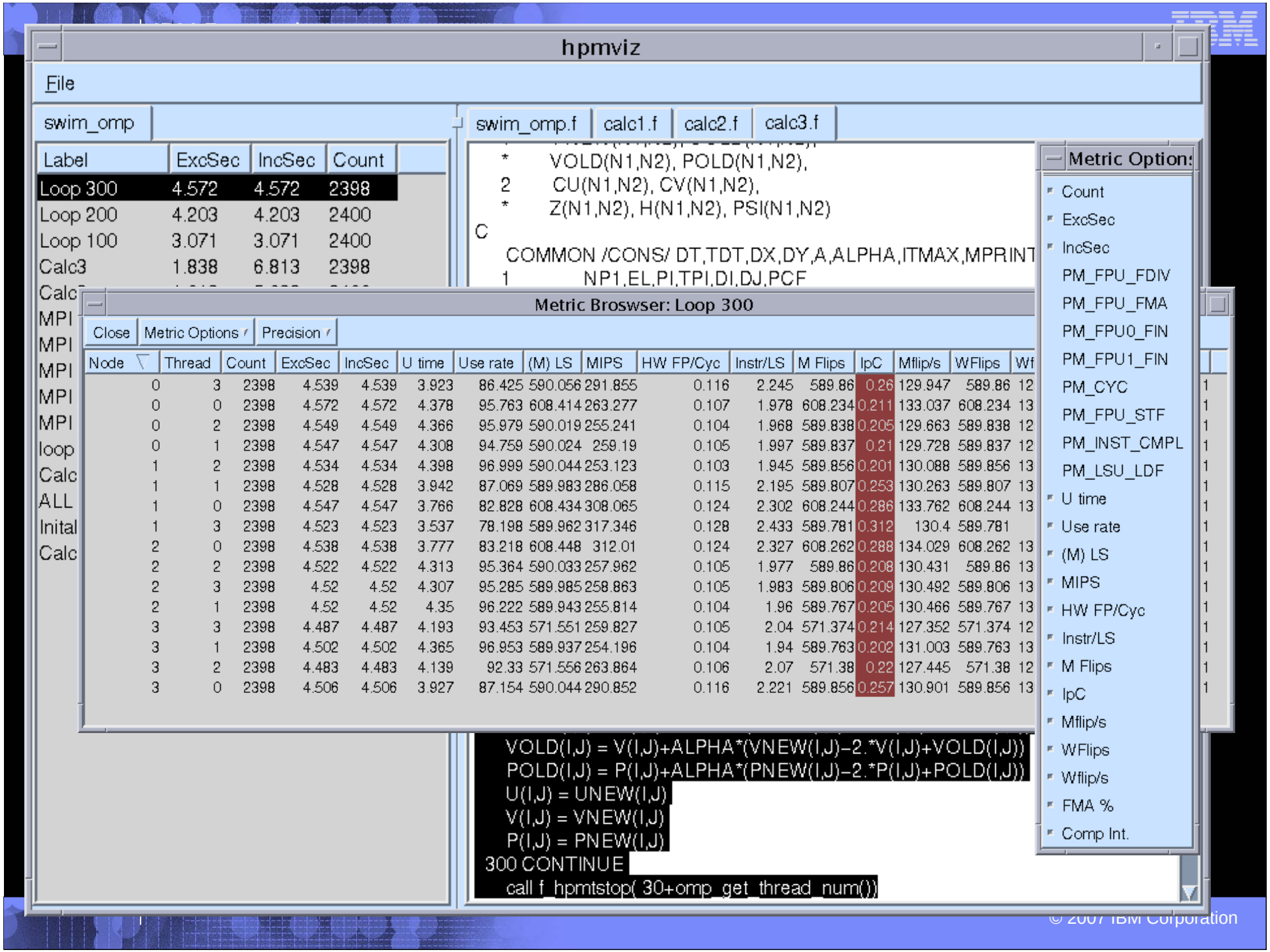
- **Declaration:**
- **Use:**
  - #include f_hpm.h

```
call f_hpminit( 0, "prog" )
call f_hpmstart( 1, "work" )
do
  call do_work()
  call f_hpmstart( 22, "more work" )
  –        call compute_meaning_of_life()
  call f_hpmstop( 22 )
end do
call f_hpmstop( 1 )
call f_hpmterminate( 0 )
```

Use MPI taskID with MPI programs

HPM: Multi-thread Support





MPI Profiler/Tracer

Message-Passing Performance

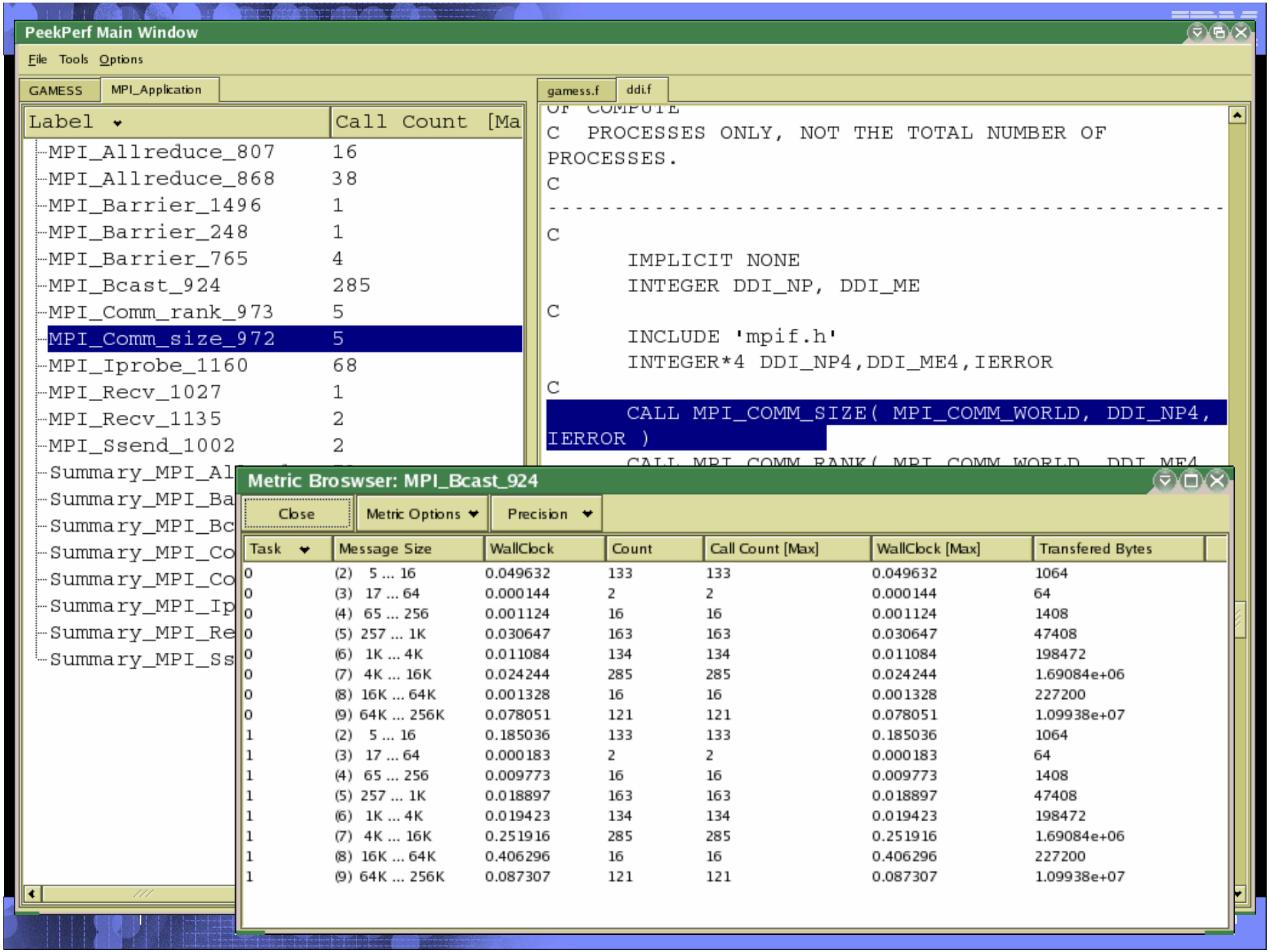
MP_Profiler

- Implements wrappers around MPI calls using the PMPI interface
  - start timer
  - call pmpi equivalent function
  - stop timer
- Captures “summary” data for MPI calls with source code traceback
- No changes to source code, but MUST compile with -g
- ~1.7 microsecond overhead per MPI call
- Does not synchronize MPI calls
- Compile with –g and link with libmpitrace.a
- Generate XML files for peekperf

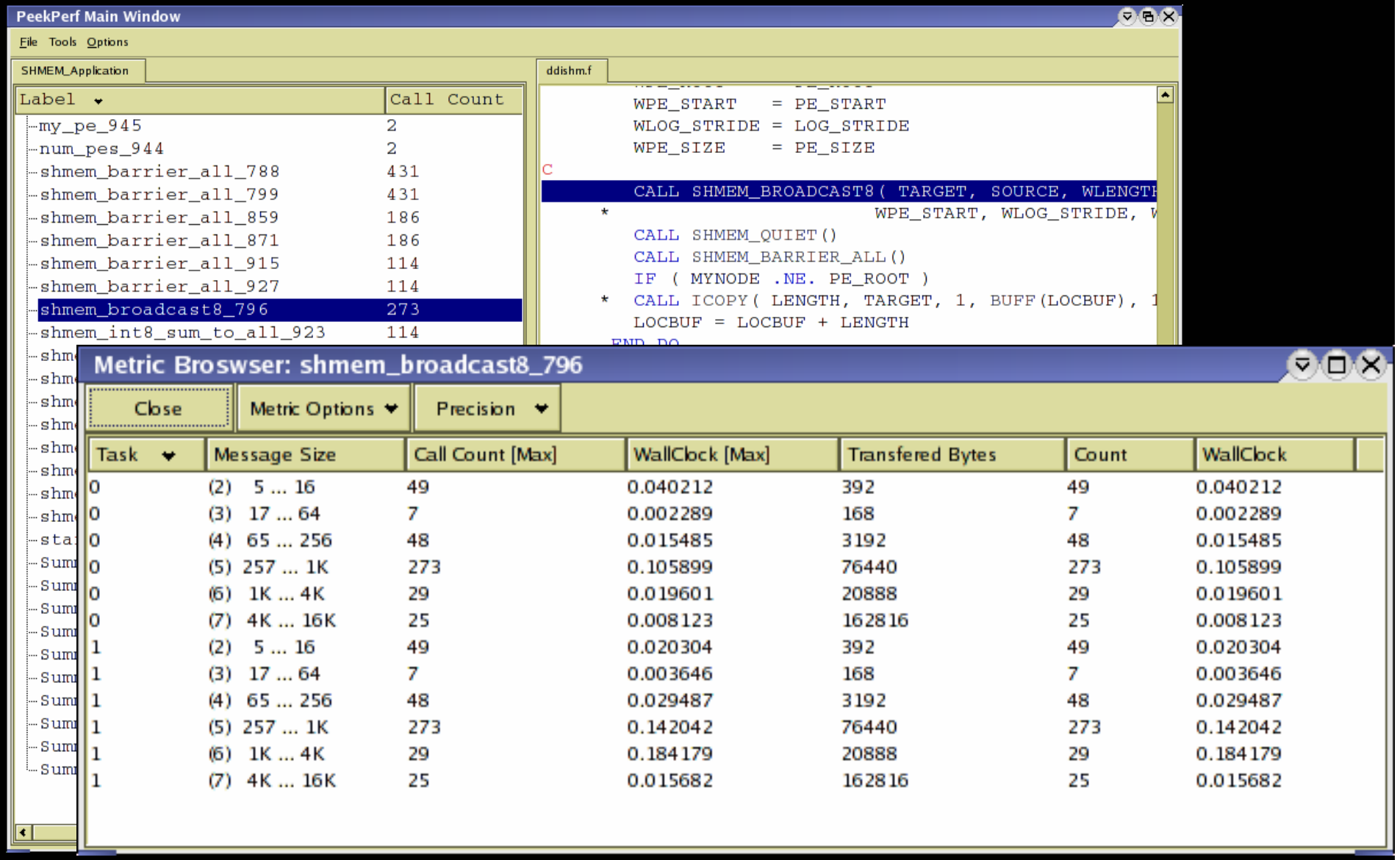
Message-Passing Performance

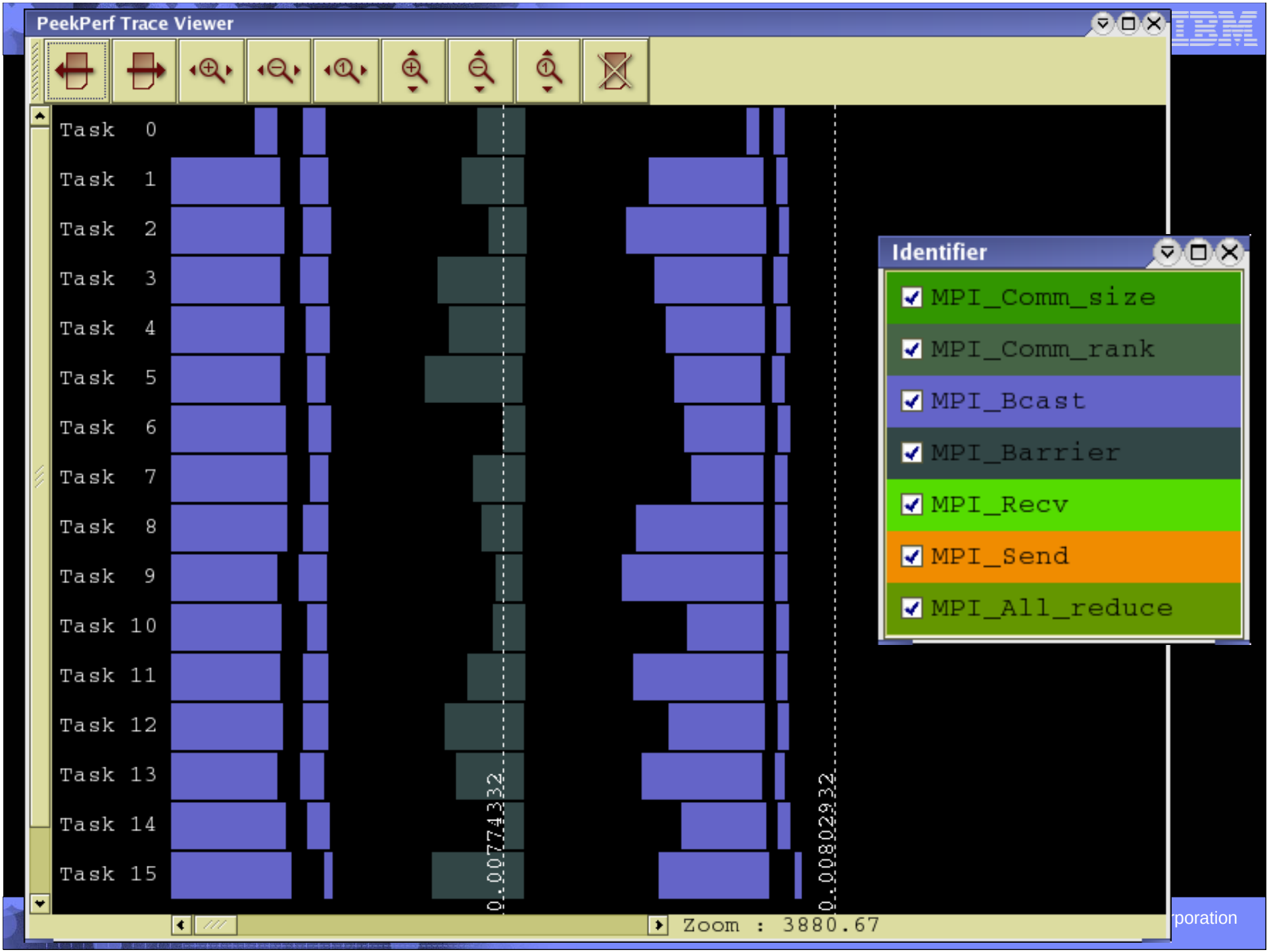
MPI Tracer

- Captures “timestamped” data for MPI calls with source traceback
- Provides a color-coded trace of execution
- ~1.4 microsecond overhead per MPI call
- Very useful to identify load-balancing issues


PeekPerf Main Window
File Tools Options
GAMESS
MPI_Application
Label
Call Count [Ma
MPI_Allreduce_807 16
MPI_Allreduce_868 38
MPI_Barrier_1496 1
MPI_Barrier_248 1
MPI_Barrier_765 4
MPI_Bcast_924 285
MPI_Comm_rank_973 5
MPI_Comm_size_972 5
MPI_Iprobe_1160 68
MPI_Recv_1027 1
MPI_Recv_1135 2
MPI_Ssend_1002 2
gamess.f
ddi.f
C PROCESSES ONLY, NOT THE TOTAL NUMBER OF
PROCESSES.
C
C
IMPLICIT NONE
INTEGER DDI_NP, DDI_ME
C
INCLUDE 'mpif.h'
INTEGER*4 DDI_NP4,DDI_ME4,IERROR
C
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, DDI_NP4,
IERROR)
Metric Broswser: MPI_Bcast_924
Close
Metric Options
Precision
Task
Message Size
WallClock
Count
Call Count [Max]
WallClock [Max]
Transfered Bytes
0 (2) 5 ... 16 0.049632 133 133 0.049632 1064
0 (3) 17 ... 64 0.000144 2 2 0.000144 64
0 (4) 65 ... 256 0.001124 16 16 0.001124 1408
0 (5) 257 ... 1K 0.030647 163 163 0.030647 47408
0 (6) 1K ... 4K 0.011084 134 134 0.011084 198472
0 (7) 4K ... 16K 0.024244 285 285 0.024244 1.69084e+06
0 (8) 16K ... 64K 0.001328 16 16 0.001328 227200
0 (9) 64K ... 256K 0.078051 121 121 0.078051 1.09938e+07
1 (2) 5 ... 16 0.185036 133 133 0.185036 1064
1 (3) 17 ... 64 0.000183 2 2 0.000183 64
1 (4) 65 ... 256 0.009773 16 16 0.009773 1408
1 (5) 257 ... 1K 0.018897 163 163 0.018897 47408
1 (6) 1K ... 4K 0.019423 134 134 0.019423 198472
1 (7) 4K ... 16K 0.251916 285 285 0.251916 1.69084e+06
1 (8) 16K ... 64K 0.406296 16 16 0.406296 227200
1 (9) 64K ... 256K 0.087307 121 121 0.087307 1.09938e+07

SHMEM Profiling Capability




PeekPerf Trace Viewer
Task 0
Task 1
Task 2
Task 3
Task 4
Task 5
Task 6
Task 7
Task 8
Task 9
Task 10
Task 11
Task 12
Task 13
Task 14
Task 15
0.007743332
0.008029332
Zoom : 3880.67
Identifier
MPI_Comm_size
MPI_Comm_rank
MPI_Bcast
MPI_Barrier
MPI_Recv
MPI_Send
MPI_All_reduce

PompProf

“Standard” OpenMP Monitoring API?

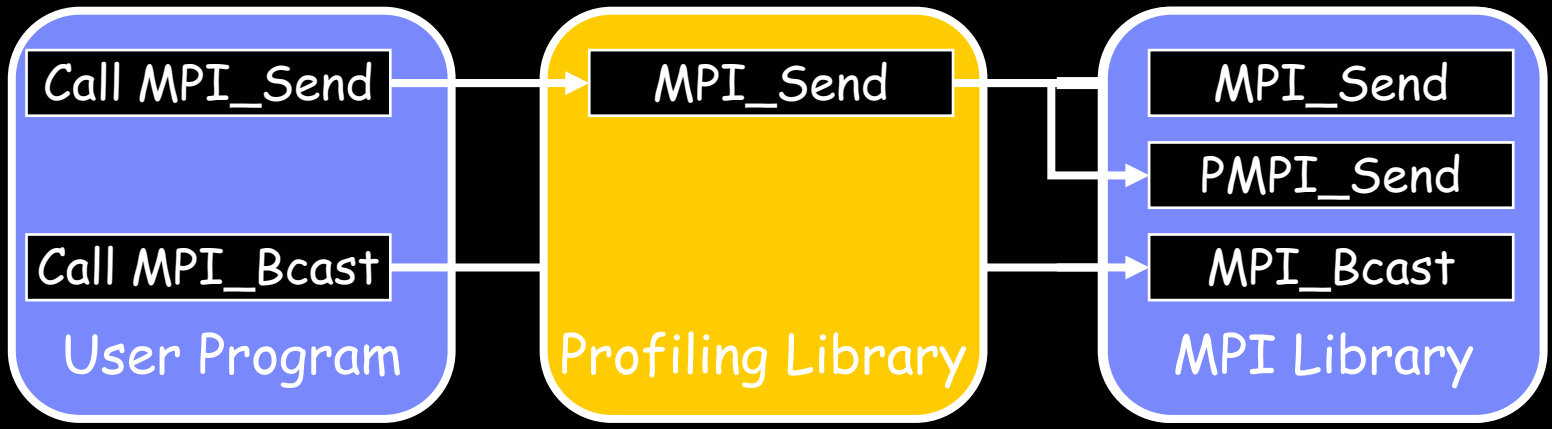
- **Problem:**
  - OpenMP (unlike MPI) does not define standard monitoring interface (at SC06 they accepted a proposal from SUN and others)
  - OpenMP is defined mainly by directives/pragmas
- **Solution:**
  - POMP: OpenMP Monitoring Interface
  - Joint Development

    - Forschungszentrum Jülich
    - University of Oregon
  - Presented at EWOMP’01, LACSI’01 and SC’01
    - “The Journal of Supercomputing”, 23, Aug. 2002.

Profiling of OpenMP Applications: POMP

- Portable cross-platform/cross-language API to simplify the design and implementation of OpenMP tools

- POMP was motivated by the MPI profiling interface (PMPI)
  - PMPI allows selective replacement of MPI routines at link time
  - Used by most MPI performance tools (including MPI Profiler/Tracer)



POMP Proposal

- **Three groups of events**
  - OpenMP constructs and directives/pragmas
    - Enter/Exit around each OpenMP construct
      - Begin/End around associated body
    - Special case for parallel loops:
      - ChunkBegin/End, IterBegin/End, or IterEvent instead of Begin/End
    - “Single” events for small constructs like atomic or flush
  - OpenMP API calls
    - Enter/Exit events around omp_set_*_lock() functions
    - “single” events for all API functions
  - User functions and regions
- **Allows application programmers to specify and control amount of instrumentation**

Example: POMP Instrumentation

```
 1:  int main() {
 2:     int id;
***     POMP_Init();
 3:
***     { POMP_handle_t pomp_hd1 = 0;
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_enter(&pomp_hd1, pomp_tid, -1, 1,
***           "49*type=pregion*file=demo.c*slines=4,4*elines=8,8**");
 4:     #pragma omp parallel private(id)
 5:     {
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_begin(pomp_hd1, pomp_tid);
 6:       id = omp_get_thread_num();
 7:       printf("hello from %d\n", id);
***       POMP_Parallel_end(pomp_hd1, pomp_tid);
 8:     }
***       POMP_Parallel_exit(pomp_hd1, pomp_tid);
***     }
***     POMP_Finalize();
 9:  }
```

DPOMP: Dynamic Performance Monitoring Interface for OpenMP

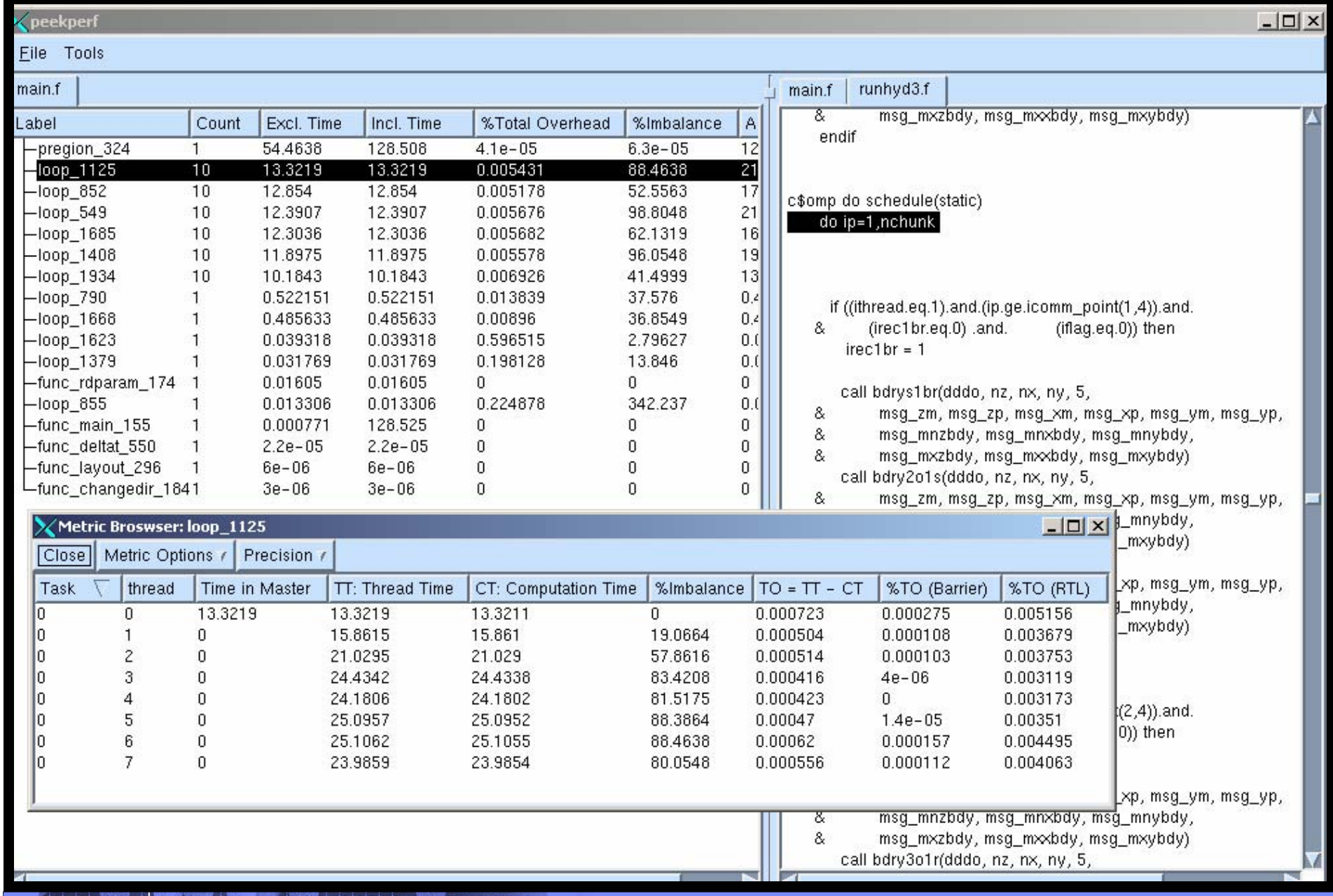
- **Collaboration with Forschungszentrum Jülich**
- **Motivation:**
  - POMP was under review (at the time)
  - May take too long to be implemented (if accepted)
- **Approach**
  - A POMP implementation based on dynamic probes
  - Built on top of DPCL
    - Modifies the binary with performance instrumentation
    - No source code or re-compilation required

SIGMA-POMP: Performance Monitoring Interface for OpenMP based on PSIGMA Instrumentation

- **Approach**
  - A POMP implementation using pSigma's binary instrumentation and rewriting
  - Built on top of pSigma
    - Modifies the binary with performance instrumentation
    - No source code or re-compilation required

POMP Profiler (PompProf)

- **Profiler for OpenMP application implemented on top of DPOMP and SIGMA-POMP**
- **Generates a detailed profile describing overheads and time spent by each thread in three key regions of the parallel application:**
  - Parallel regions
  - OpenMP loops inside a parallel region
  - User defined functions
  -
- **Profile data is presented in the form of an XML file that can be visualized with PeekPerf**



SIGMA

SIGMA Infrastructure: Highlights

Complex infrastructure: symbolic, dynamic, performance-oriented environment for binary instrumentation

- Inject arbitrary user-supplied probes into a binary application

Provides a platform to:

- Develop performance tools
- Experiment with memory performance models
- Ask “what-if” questions regarding data and code rearrangements
- Provide feedback on design of new memory architectures
- Identify performance bottlenecks due to the memory hierarchy and data layout

SIGMA Memory Profiler is integrated in the HPC Toolkit nm

Memory Profile:

Provide counters such as hits, misses, cold misses for

- each cache level
- each function
- each data structure
- each data structure within each function

Output sorted by the SIGMA memtime:

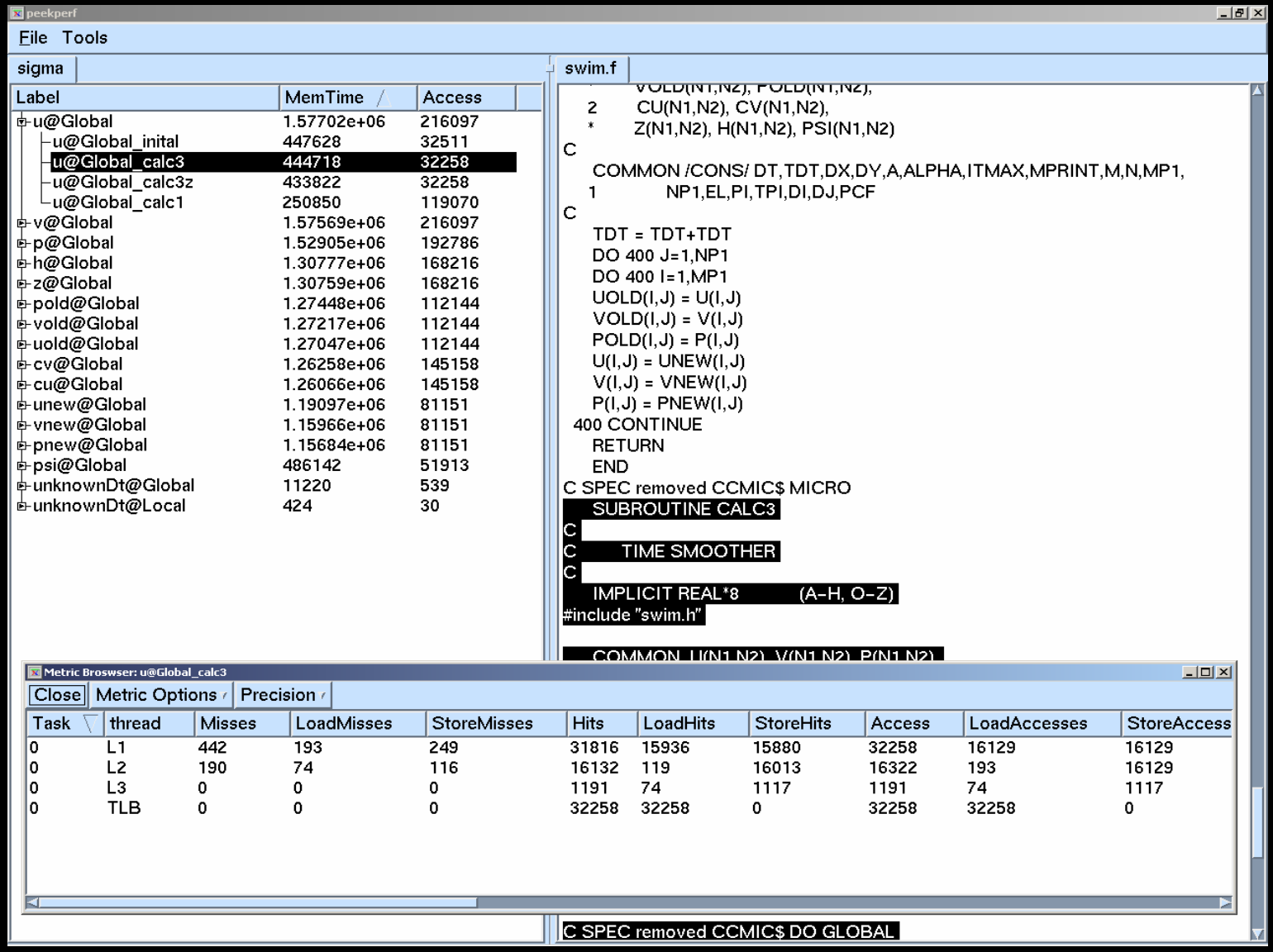
- SUM(LoadHits(i)*LoadLat(i) + StoreHits(i)*StoreLat(i)) +
- #TLBmisses * Lat(TLBmiss)

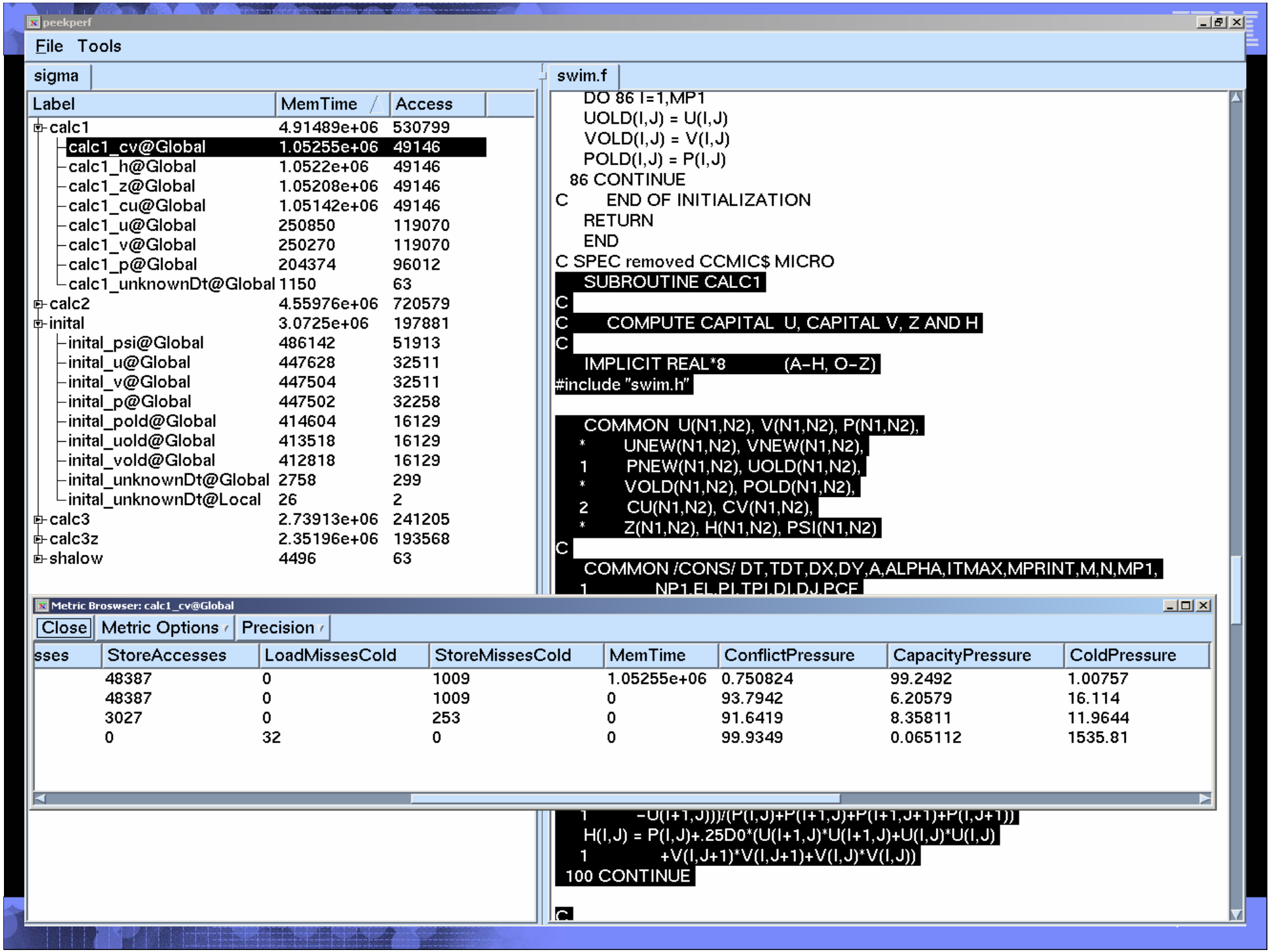
- **memtime should track wall time for memory bound applications**

Memory Profile Output

	L1	L2	L3	TLB	MEM
FUNCTION: calc1 (memtime = 0.0050)					
Load Acc/Miss/Cold	522819/2252/1	2252/345/0	345/0/0	784419/126/0	0/-/-
Load Acc/Miss Ratio	232	6	-	6225	-
Load Hit Ratio	99.57%	84.68%	100.00%	99.98%	-
Est. Load Latency	0.0008 sec	0.0000 sec	0.0000 sec	0.0001 sec	0.0000 sec
Load Traffic	-	238.38 Kb	43.12 Kb	-	0.00 Kb
……					
FUNCTION: calc2 (memtime = 0.0042)					
Load Acc/Miss/Cold	622230/2631/0	2631/1661/0	1661/0/0	814269/94/0	0/-/-
Load Acc/Miss Ratio	236	1	-	8662	-
Load Hit Ratio	99.58%	36.87%	100.00%	99.99%	-
Est. Load Latency	0.0010 sec	0.0000 sec	0.0001 sec	0.0001 sec	0.0000 sec
Load Traffic	-	121.25 Kb	207.62 Kb	-	0.00 Kb
……					
	L1	**L2**	**L3**	**TLB**	**MEM**
DATA: u (memtime = 0.0012)					
Load Acc/Miss/Cold	167710/708/0	708/317/0	317/0/0	216097/31/0	0/-/-
Load Acc/Miss Ratio	236	2	-	6970	-
Load Hit Ratio	99.58%	55.23%	100.00%	99.99%	-
Est. Load Latency	0.0003 sec	0.0000 sec	0.0000 sec	0.0000 sec	0.0000 sec
Load Traffic	-	48.88 Kb	39.62 Kb	-	0.00 Kb
……					
DATA: v (memtime = 0.0012)					
Load Acc/Miss/Cold	167710/721/0	721/316/0	316/0/0	216097/31/0	0/-/-
Load Acc/Miss Ratio	232	2	-	6970	-
Load Hit Ratio	99.57%	56.17%	100.00%	99.99%	-
Est. Load Latency	0.0003 sec	0.0000 sec	0.0000 sec	0.0000 sec	0.0000 sec
Load Traffic	-	50.62 Kb	39.50 Kb	-	0.00 Kb
……					

Memory Profile Viewer – Data Structure Focus




peekperf
File Tools
sigma
swim.f
Label
MemTime
Access
calc1 4.91489e+06 530799
calc1_cv@Global 1.05255e+06 49146
calc1_h@Global 1.0522e+06 49146
calc1_z@Global 1.05208e+06 49146
calc1_cu@Global 1.05142e+06 49146
calc1_u@Global 250850 119070
calc1_v@Global 250270 119070
calc1_p@Global 204374 96012
calc1_unknownDt@Global 1150 63
calc2 4.55976e+06 720579
inital 3.0725e+06 197881
inital_psi@Global 486142 51913
inital_u@Global 447628 32511
inital_v@Global 447504 32511
inital_p@Global 447502 32258
inital_pold@Global 414604 16129
inital_uold@Global 413518 16129
inital_vold@Global 412818 16129
inital_unknownDt@Global 2758 299
inital_unknownDt@Local 26 2
calc3 2.73913e+06 241205
calc3z 2.35196e+06 193568
shalow 4496 63
DO 86 I=1,MP1
UOLD(I,J) = U(I,J)
VOLD(I,J) = V(I,J)
POLD(I,J) = P(I,J)
86 CONTINUE
C END OF INITIALIZATION
RETURN
END
C SPEC removed CCMIC$ MICRO
SUBROUTINE CALC1
C
C COMPUTE CAPITAL U, CAPITAL V, Z AND H
C
IMPLICIT REAL*8 (A-H, O-Z)
#include "swim.h"
COMMON U(N1,N2), V(N1,N2), P(N1,N2),
* UNEW(N1,N2), VNEW(N1,N2),
1 PNEW(N1,N2), UOLD(N1,N2),
* VOLD(N1,N2), POLD(N1,N2),
2 CU(N1,N2), CV(N1,N2),
* Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT,TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,M,N,MP1,
1 NP1,EL,PI,TPI,DI,DJ,PCF
Metric Broswer: calc1_cv@Global
Close
Metric Options
Precision
sses
StoreAccesses
LoadMissesCold
StoreMissesCold
MemTime
ConflictPressure
CapacityPressure
ColdPressure
48387 0 1009 1.05255e+06 0.750824 99.2492 1.00757
48387 0 1009 0 93.7942 6.20579 16.114
3027 0 253 0 91.6419 8.35811 11.9644
0 32 0 0 99.9349 0.065112 1535.81
1 -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
H(I,J) = P(I,J)+.25D0*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
1 +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
100 CONTINUE
C

Usage:

- Binary instrumentation:
  - sigmaInst –d appbin (produces executable appbin.inst)
  - ./appbin.inst [app-args] –sigma sigmaArchFile

Things to be aware of:

- It's a simulator!! (may not want to use on long-running apps)
- Overhead is about 2 to 3 orders of magnitude slower
- Only functions statically linked in the executable can be instrumented
- Application must be compiled with -g

How to overcome the speed limitations of the simulator? Partial Instrumentation

- Statically select functions to instrument:
  - sigmaInst –d –dfunc f1,f2,…,fn appbin
- Dynamically select code sections to instrument:
  - #include "signal_sigma.h“

```
for (lp=1; lp<NIT; lp++) {

   /* start sigma */
   if ((lp == 2) || (lp==7)) {
      signal_sigma(TRACE_ON, lp);
   }

   for ( i=0; i<n; i++ ) {
      for ( k=0; k< n; k++ ) {
         u[n * i + k] = 0.0;
         f[n * i + k] = rhs;
      }
   }

   /* stop sigma */
   if ((lp == 2) || (lp==7)) {
      signal_sigma(TRACE_OFF, lp);
   }
}
```

Use partial instrumentation with a grain of salt!

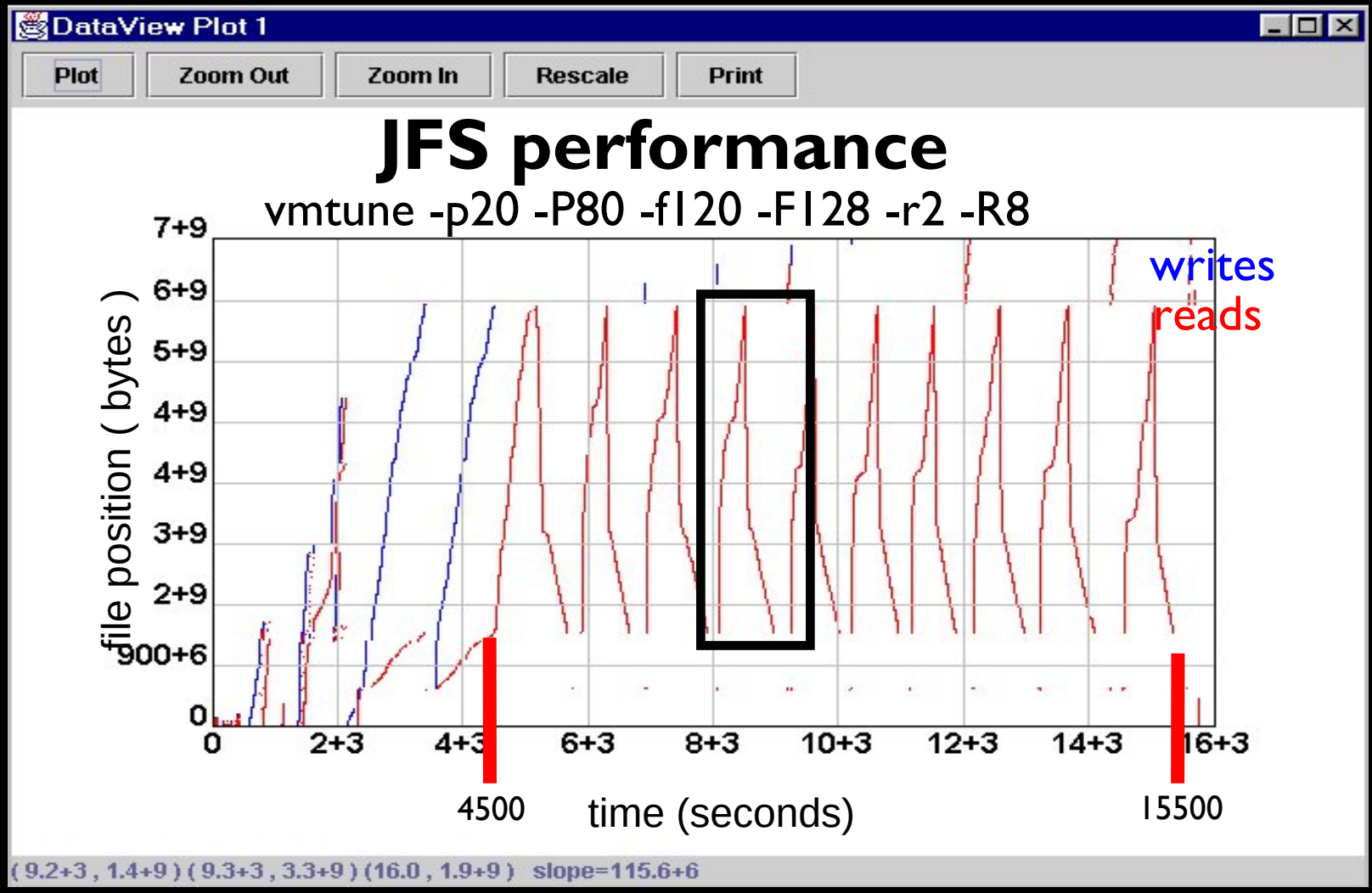
- Unless the entire application is instrumented, there will be approximation in the results (cache state won't be accurate between activation/deactivation)
- Use dynamic selection at phase or loop boundaries or when the cache can be assumed 'cold'
- Use sigma signals to reset the cache
- Ongoing research projects to find optimal sampling techniques (collaboration with UMD, SDSC)
- Ongoing research on automatic sampling

MIO

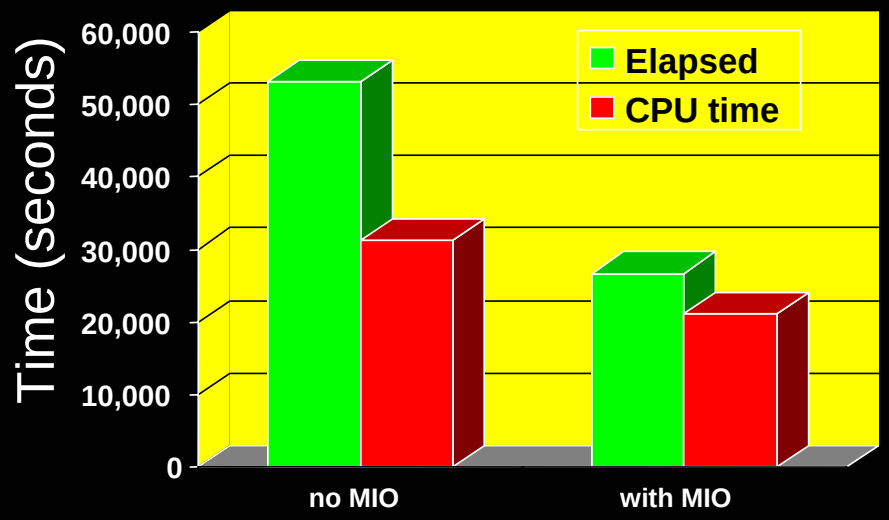
Modular I/O Performance Tool (MIO)

- **I/O Analysis**
  - Trace module
  - Summary of File I/O Activity + Binary Events File
  - Low CPU overhead
- **I/O Performance Enhancement Library**
  - Prefetch module (optimizes asynchronous prefetch and write-behind)
  - System Buffer Bypass capability
  - User controlled pages (size and number)
- **Recoverable Error Handling**
  - Recover module (monitors return values and errnor + reissues failed requests)
- **Remote Data Server**
  - Remote module (simple socket protocol for moving data)
- **Shared object library for AIX**

Performance Visualization (work in progress)



MSC.Nastran V2001



Benchmark:
SOL 111, 1.7M DOF, 1578 modes, 146 frequencies, residual flexibility and acoustics. 120 GB of disk space.

Machine:
4-way, 1.3 GHz p655, 32 GB with 16 GB large pages, JFS striped on 16 SCSI disks.

MSC.Nastran:
V2001.0.9 with large pages, dmp=2 parallel=2 mem=700mb
The run with MIO used mio=1000mb

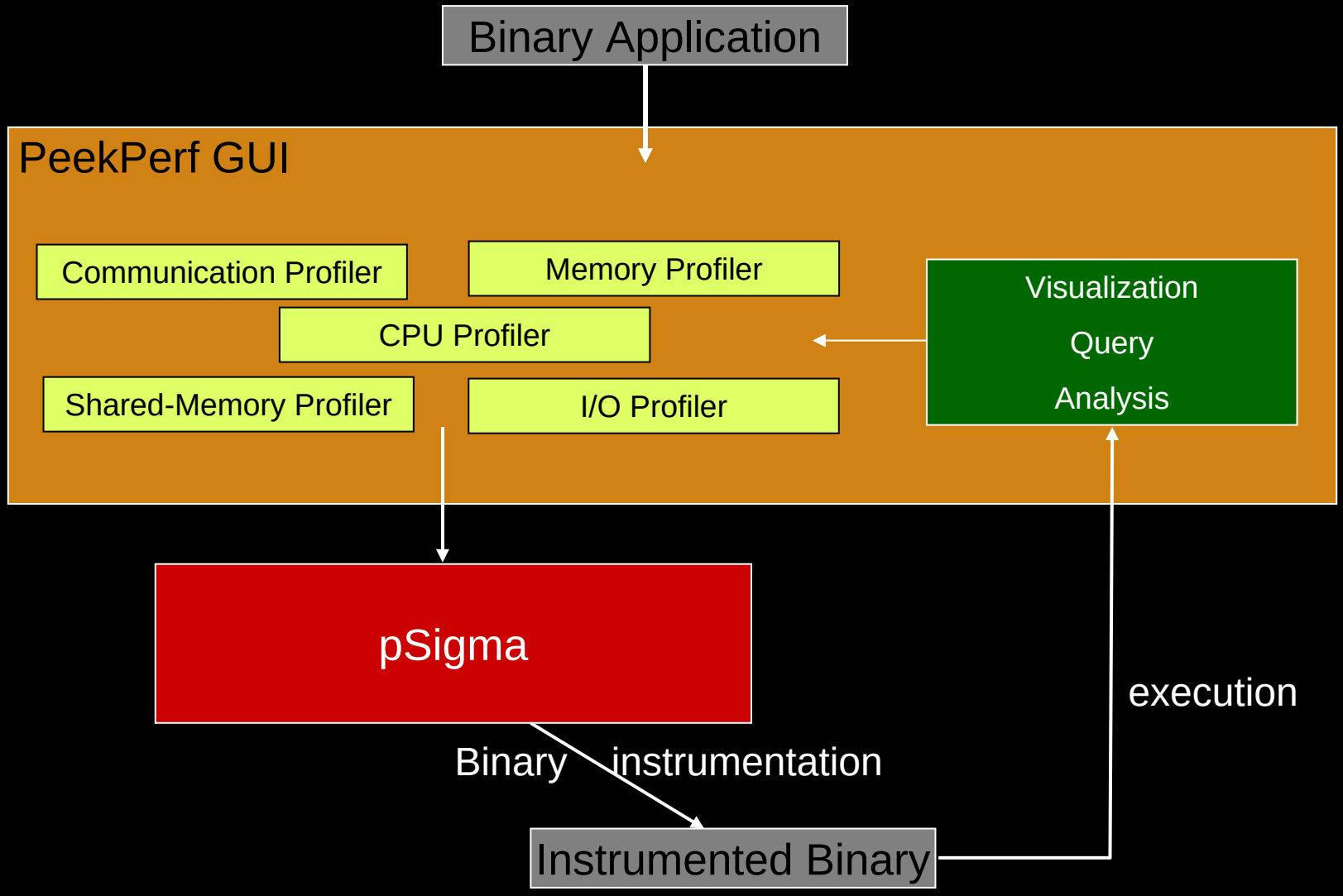
6.8 TB of I/O in 26666 seconds is an average of about 250 MB/sec

The IBM HPC Toolkit

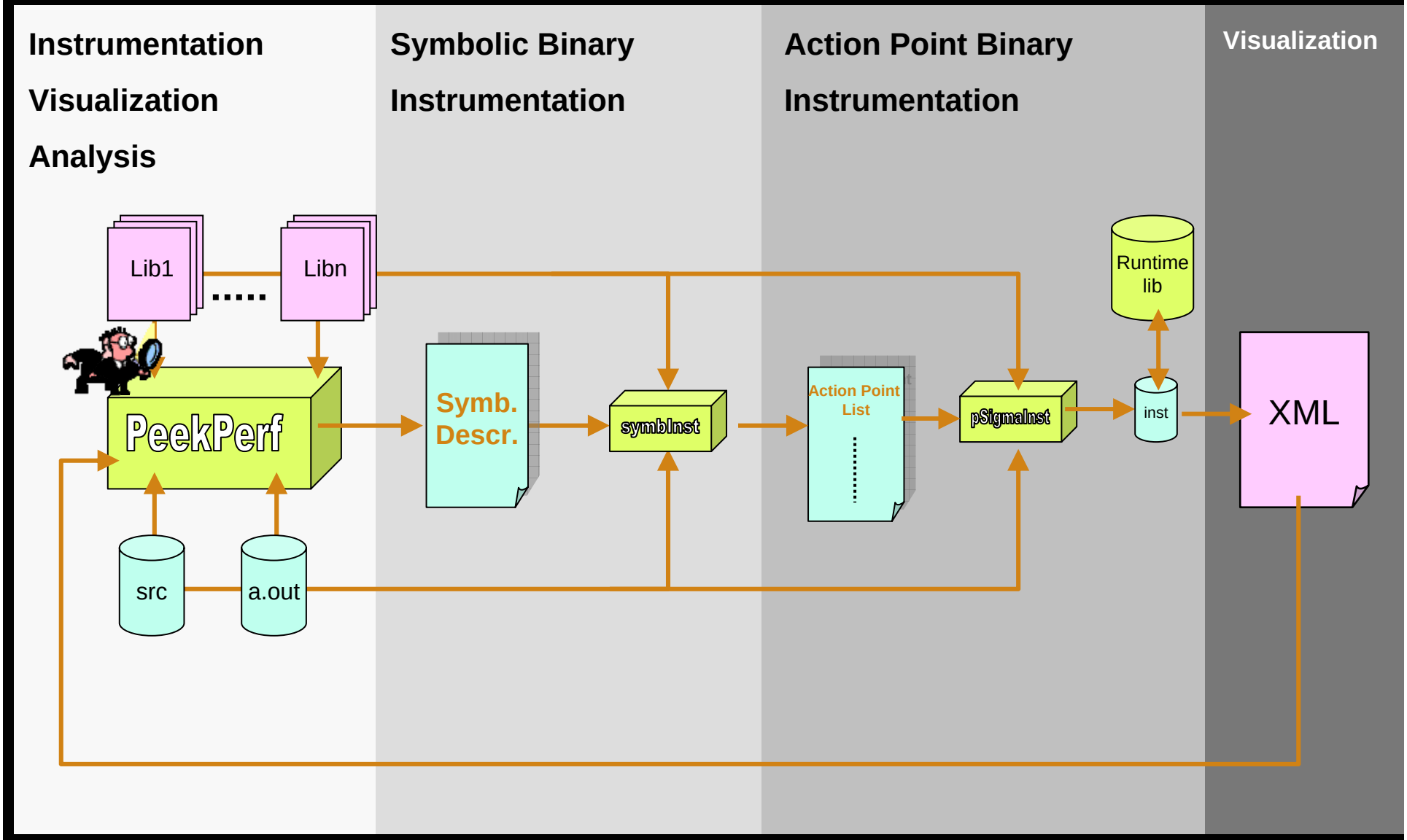
IBM HPC Toolkit

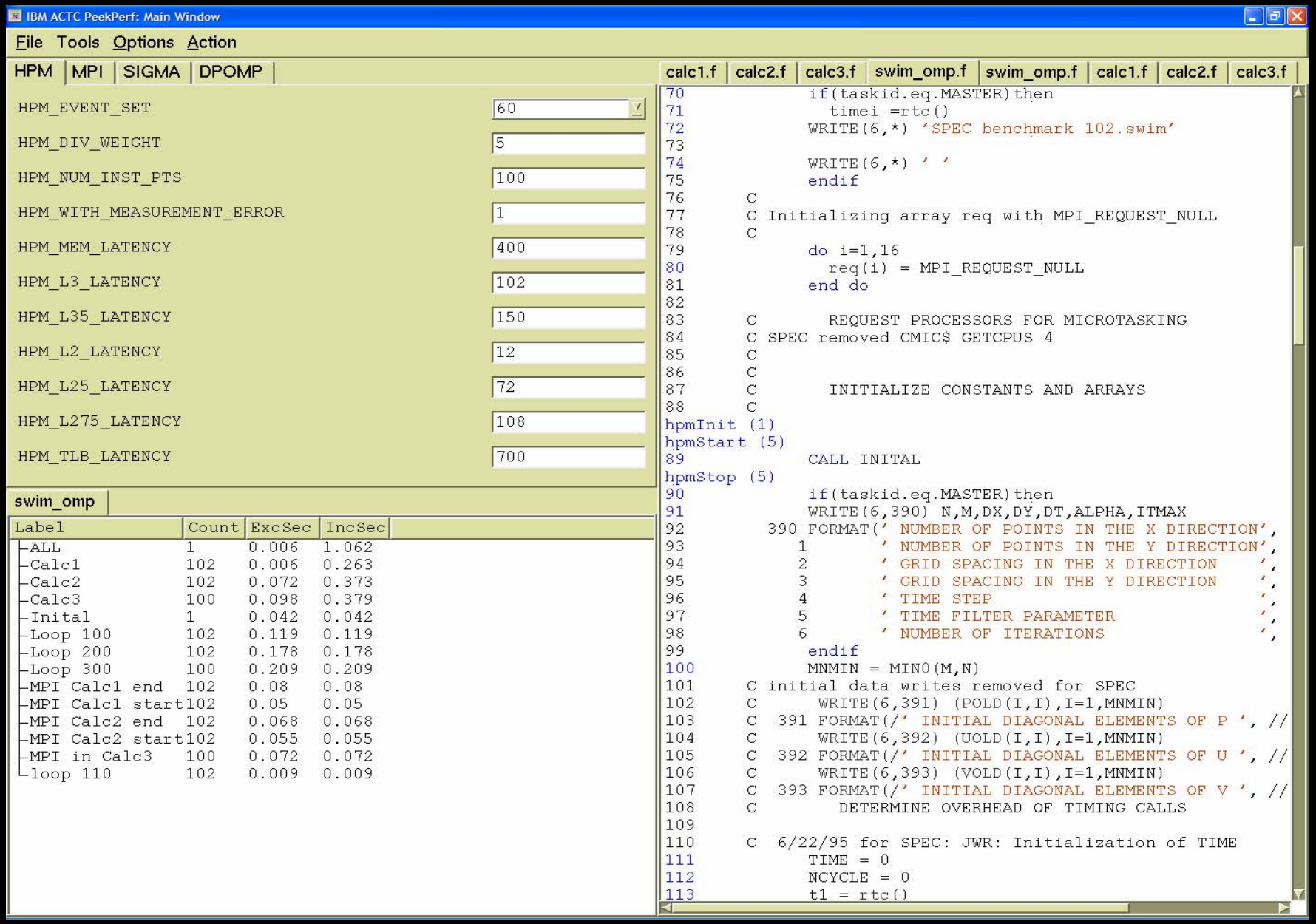
- Instrumentation at the binary level
- Centralized GUI for instrumentation and analysis
- Dynamic instrumentation capabilities
- Enhanced with graphics capabilities (bar charts, plots etc)
- Simultaneous instrumentation for all tools (one run!)
- Query capabilities: compute derived metrics and plot them
- Selective instrumentation of MPI functions

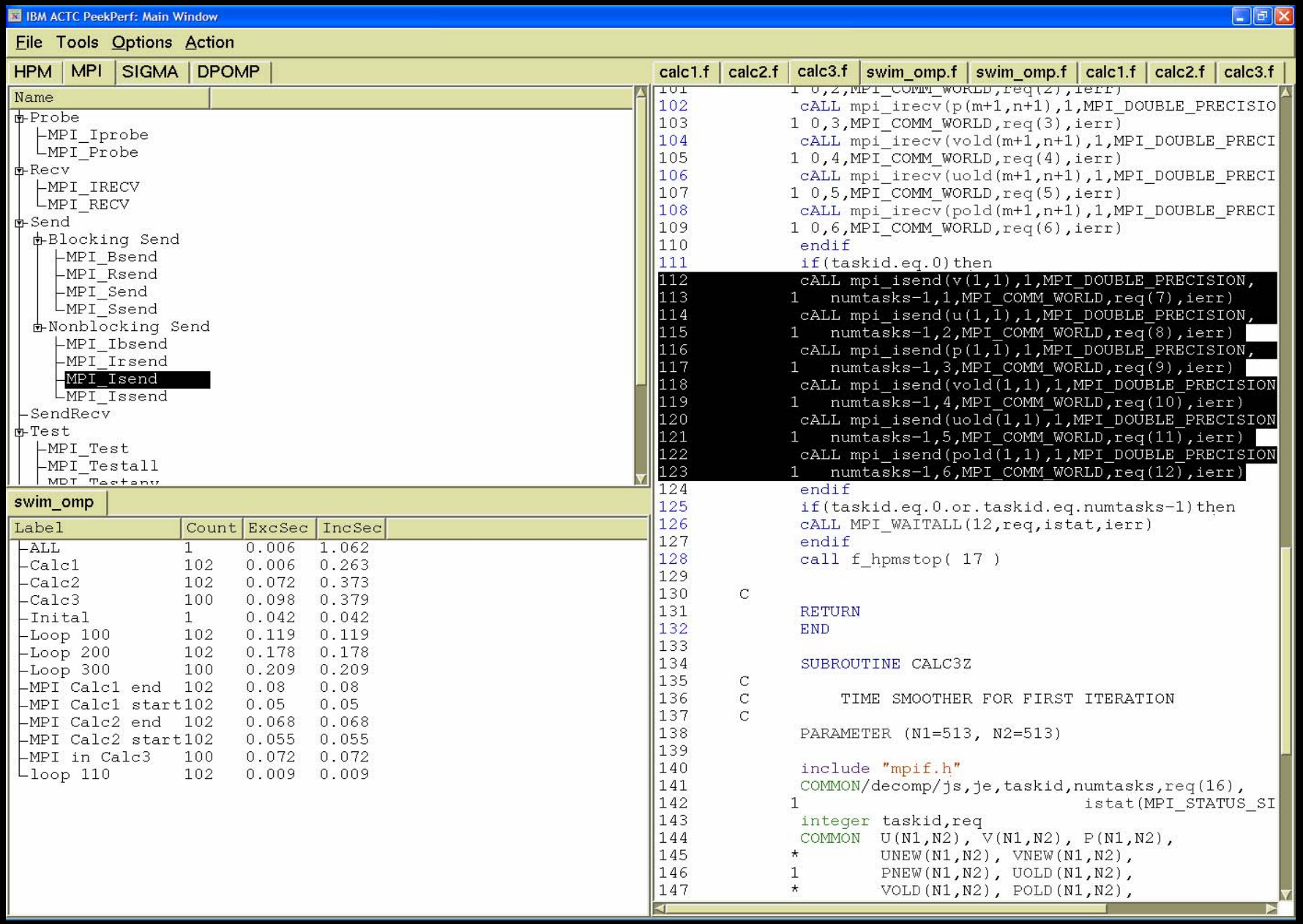
Structure of the IBM HPC toolkit



PeekPerf: Graphical Instrumentation, Visualization and Analysis







Summary

- **The IBM HPC Toolkit is the IBM environment for HPC application performance analysis**
  - Years of field testing and experience
  - Visual source code traceback for all metrics
  - Common framework for simultaneous performance analysis of CPU, MPI, OpenMP, Memory and I/O
  - Available across IBM HPC servers (pSeries, Linux, Blue Gene)
- **Future infrastructure characteristics**
  - Instrumentation of the binary
  - Common instrumentation and analysis GUI
  - Dynamic instrumentation capabilities

Questions / Comments